Fast SVM Regularization Path Computation - Theory and Implementation

D I P L O M A T H E S I S

Awarding the Academic Degree
Diplom-Informatiker

FRIEDRICH-SCHILLER-UNIVERSITY JENA
Faculty for Mathematics and Computer Science
Adviser: Prof. Dr. Joachim Giesen
Submitted by Torsten Welsch
Born on August 15, 1986 in Jena

Jena, March 31, 2011

1 Abstract

Im Fokus dieser Diplomarbeit steht die Implementierung und theoretische Analyse eines neuen Algorithmus zur Berechnung des gesamten Lösungspfades von allgemeinen Support Vector Machines bezüglich ihres Regularisierungsparameters. Dessen optimaler Wert wird im Ergebnis durch einen solchen Pfad deutlich leichter auffindbar. Erreicht wird dieses Ziel mit Hilfe der nicht-approximativen Criss-Cross Methode aus dem Bereich der linearen Komplementaritätsprobleme. Neben dem geometrischen Verhalten dieser Methode wird insbesondere auf deren effiziente Initialisierung zu Beginn eines Lösungspfades eingegangen. Darüber hinaus zeigt diese Arbeit auf, dass auch Probleme der Conjoint Analyse in Support Vector Machines überführt und entsprechend gelöst werden können. Abschließend werden die theoretischen Resultate anhand von Conjoint-Analyse-Datensätzen und solchen für Support Vector Machines veranschaulicht.

2 Contents

1 Abstract **2**

2 Contents **3**

3 Preamble **5**

4 Theory **6**
4.1 The Criss-Cross Method 6
4.1.1 Parameterized Quadratic Programs 7
4.1.2 The Parametric Linear Complementarity Problem 8
4.1.3 The Criss-Cross Algorithm 9
4.1.4 Varying the Regularization Parameter 11
4.1.5 Odds and Ends 12
4.2 Support Vector Machines 12
4.2.1 Basic Concepts 13
4.2.2 Primal SVM 14
4.2.3 Lagrange Duality 15
4.2.4 Dual SVM 17
4.2.5 Kernels 18
4.3 SVM to pQP 22
4.4 Basis Reconstruction 25
4.4.1 General Case 25
4.4.2 Determining the Path's Start 28
4.4.3 Special Case 29
4.4.4 Artificial Basis Derivation 31
4.5 Conjoint Analysis 37
4.5.1 Introduction 37
4.5.2 Problem Setup 37
4.5.3 SVMs and Conjoint Analysis 39
4.5.4 Part Worth Values' Computation 40
4.6 Understanding the Geometry 40

5 Implementation **43**
5.1 Data Structures 43
5.1.1 M's Storage 43
5.1.2 M_B's Rowwise Storage 43
5.1.3 M_B's Columnwise Storage 44
5.1.4 Pivot Selection Arrays 44
5.1.5 Active Submatrix Flag 45
5.2 Common Techniques 45
5.2.1 LU Decomposition 45

5.2.2 Solving Systems of Linear Equations . . . 48
5.2.3 Iterative Refinement . . . 49
5.2.4 Diagonal and Exchange Pivot . . . 50
5.3 Accuracy Hot Spots . . . 51
5.3.1 Criss-Cross Algorithm . . . 52
5.3.2 Varying the Regularization Parameter . . . 55
5.3.3 Basis Reconstruction - General Case . . . 57
5.4 Data Formats . . . 57
5.5 Further Implementation Aspects . . . 59
5.5.1 Parallel Computing . . . 59
5.5.2 Sherman-Morrison Formula . . . 59
5.5.3 The General Case . . . 60
5.5.4 Handling CPLEX and LIBSVM . . . 60
5.6 Experimental Results . . . 63
5.6.1 Introduction . . . 63
5.6.2 Engine Linear Kernel . . . 65
5.6.3 Engine Gaussian Kernel . . . 68
5.6.4 CeBIT1 Linear Kernel . . . 70
5.6.5 CeBIT1 Gaussian Kernel . . . 73
5.6.6 CeBIT2 Linear Kernel . . . 75
5.6.7 CeBIT2 Gaussian Kernel . . . 78
5.6.8 Adult Linear Kernel . . . 80
5.6.9 Adult Gaussian Kernel . . . 82
5.6.10 Splice Linear Kernel . . . 84
5.6.11 Splice Gaussian Kernel . . . 86
5.6.12 Simple . . . 88

6 Statement of Independence **93**

7 References **93**

8 List of Figures **94**

9 List of Tables **96**

3 Preamble

The diploma thesis at hand is the result of the my final year project at the Friedrich-Schiller-University Jena under the guidance of Prof. Dr. Joachim Giesen. During this work, a fast generic algorithm being able to compute the entire regularization path of any support vector machine has been implemented and theoretically analyzed. Hence, this work is subdivided into a theoretical and a practical part:

In the theory section, we first introduce the concepts of parameterized quadratic programming and support vector machines. Afterwards, we describe an exact solver for parameterized linear complementarity problems - the criss-cross method. In this context, we demonstrate how to convert any soft margin support vector machine into a linear complementarity problem and discuss different approaches to speed up the computation of the regularization path using the criss-cross method. In the end of this section, we introduce the concept of conjoint analysis and describe how such problems can be transformed into support vector machines as well. Finally, we demonstrate the geometrical characteristics of the criss-cross method.

In the practical part of this work, the implementation of our algorithm is described. Starting with the data structure and the description of common algorithmical techniques, the section's focus lies on accuracy hot spots as the criss-cross method is non-approximate. Finally, some notes about the used data formats and further implementation aspects are followed by the large subsection of experimental results. Here, we test our implementation and the theory with different conjoint analysis and support vector machine data sets.

Firstly, I am indebted to Prof. Dr. Joachim Giesen who supported this work in every way needed. Secondly, my gratitude goes to my wife Magdalena Welsch for proofreading this diploma thesis. And last but not least, I hope that my son Simon will forgive me for working all the time he has constantly been demanding attention.

Weimar, March 2011

Torsten Welsch

4 Theory

4.1 The Criss-Cross Method

This work starts with the description of a new generic algorithm using the simple and elegant non-approximate **criss-cross method**. First, some definitions are stated. The first two are taken from [JMT04, p.49] and the last is correspondingly[1] taken from [LS90, p.38].

Definition: A subset $K \subset \mathbb{R}^n$ is called **convex** if for all $x, y \in K$ and $\eta \in (0,1)$ the point $(1-\eta)x + \eta y$ belongs to K.

Definition: Let $K \subset \mathbb{R}^n$ be convex and $f : K \to \mathbb{R}$. The function f is called **convex** if for all $x, y \in K$ and $\eta \in (0,1)$ the following inequality holds:

$$f\left(\eta x + (1-\eta)\, y\right) \leq \eta f\left(x\right) + (1-\eta)\, f\left(y\right). \tag{1}$$

Definition: A **quadratic program** (QP) requires the minimization (or maximization) of a quadratic objective function subject to equality or inequality constraints in which the constraint functions are linear. [...] Typically, a QP may be presented in the manner

$$\begin{array}{lrl} \textbf{QP} & \min_{x \in K} & \frac{1}{2} x^T Q x + c^T x \\ & \text{s.t.} & Ax \geq b \\ & & x \geq 0\ . \end{array}$$

There are m linear main constraints $Ax \geq b$ in n non-negative variables $x \in K$. [...] Since linear functions are both convex and concave, the convexity of the QP turns on that of the quadratic form in its objective function. If Q is positive semidefinite, that is

$$x^T Q x \geq 0$$

for all $x \in K$, then $x^T Q x$ is a convex function.

[1] No changes with regard to content; only modifications in notation and highlighting.

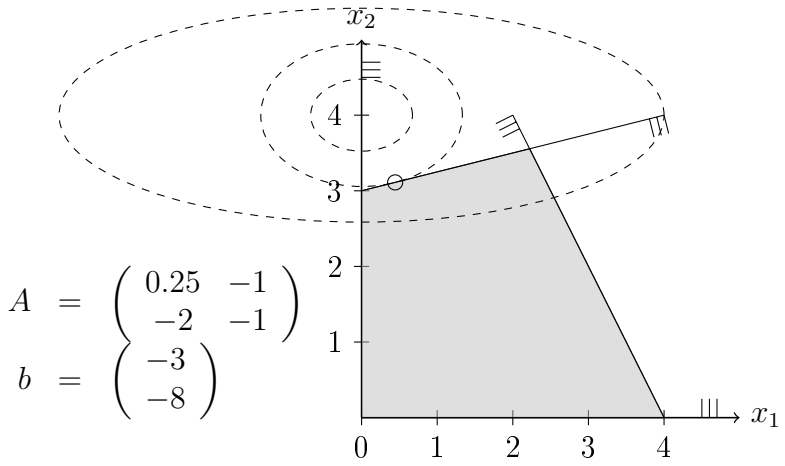

Figure 1: Simple QP example in which $\frac{1}{2}x_1^2 + x_2^2 - 8x_2$ defines the objective function and $(\frac{4}{9}, \frac{28}{9})^T$ is the optimal solution $\circ$.

4.1.1 Parameterized Quadratic Programs

Here, the interest lies in parameterized quadratic programs (pQPs) of the form

$$\boxed{\begin{array}{lll} \textbf{pQP} & \min_{x \in \mathbb{R}^n} & \frac{1}{2}x^T Q x + c\,(\mu)^T\, x \\ & \text{s.t.} & A x \geq b\,(\mu) \\ & & x \geq 0\ , \end{array}} \tag{2}$$

where

- Q is an $n \times n$ symmetric positive semidefinite (PSD) matrix (quadratic term of the objective function),
- c is an n-vector (linear term of the objective function),
- A is an $m \times n$ matrix (constraint matrix) and
- b is an m-vector (right-hand side of the constraints).

Furthermore, $c : \mathbb{R} \to \mathbb{R}^n$ and $b : \mathbb{R} \to \mathbb{R}^m$ are functions that describe how the linear term of the objective function and the right-hand side of the constraints vary with some real **regularization parameter** μ.

The task of solving a pQP for all possible values of the parameter μ in a given interval $[\mu_{\min}, \mu_{\max}]$ is called **parametric quadratic programming**. For now, the assumption is that the pQP has an optimal solution for all μ in that interval. The general case is handled in (4.1.5).

The aim is to compute a **regularization path** $x^* : \mathbb{R} \to \mathbb{R}^n$ that describes the solution as a function of the parameter μ. In fact, given any value of $\mu \in [\mu_{\min}, \mu_{\max}]$, an optimal solution x^* to the pQP should be retrieved quickly without having to solve the problem from scratch again.

Note that the solution x^* is **piecewise linear** in the parameter μ if c and b are linear functions in μ, see for example [Rit62].

4.1.2 The Parametric Linear Complementarity Problem

The main idea is to transform the pQP [(2), p.7] into a **parametric linear complementarity problem** (pLCP) and then to use the criss-cross algorithm to update the solution quickly while μ varies.

In the following, the well-known **Karush-Kuhn-Tucker** (KKT) **optimality conditions** for quadratic programs are stated, see for example [CPS92, subsection 2.8].

Theorem: An n-vector x is an optimal solution to the pQP [(2), p.7] if and only if there exists an n-vector u as well as m-vectors y and v so that

$$\begin{array}{lrcl} \text{(i)} & v = Ax - b(\mu) \geq 0 & \text{and} & x \geq 0 \\ \text{(ii)} & u = c(\mu) - A^T y + Qx \geq 0 & \text{and} & y \geq 0 \\ \text{(iii)} & x^T u = 0 & \text{and} & y^T v = 0\ , \end{array} \tag{3}$$

where (i) encodes the primal feasibility of x, (ii) encodes the dual feasibility of y and (iii) refers to the complementary slackness.

The three conditions of the previous theorem (3) can be rewritten as a pLCP in the form

$$\boxed{\begin{array}{lrcl} \textbf{pLCP} & w - Mz & = & q(\mu) \\ & w, z & \geq & 0 \\ & w^T z & = & 0\ , \end{array}} \tag{4}$$

where $w = \begin{pmatrix} u \\ v \end{pmatrix}$, $z = \begin{pmatrix} x \\ y \end{pmatrix}$, $q(\mu) = \begin{pmatrix} c(\mu) \\ -b(\mu) \end{pmatrix}$ and $M = \begin{pmatrix} Q & -A^T \\ A & 0 \end{pmatrix}$.

Note that the matrix M in (4) is a PSD matrix because

$$w^T M w = \left(u^T Q + v^T A, -u^T A^T\right) w = u^T Q u + v^T A u - u^T A^T v = u^T Q u \geq 0$$

holds for any $w \in \mathbb{R}^{m+n}$. In order to solve the pQP, vectors w and z that satisfy (4) have to be found, where the first n components of the $(n+m)$-vector z form a **solution** to the pQP. The reduction of a QP to an LCP is well-known, see for example [CPS92, subsection 1.2].

4.1.3 The Criss-Cross Algorithm

The criss-cross algorithm is a combinatorial method for finding vectors w and z that satisfy the LCP [(4), p.8], where $q = q(\mu)$ is fixed for now. The case of varying μ is addressed in (4.1.4). It is guaranteed that the method terminates with a solution or a proof of infeasibility of the LCP, given that M is a **sufficient matrix**, see for example [FNT98]. This matrix class contains all PSD matrices, meaning that the criss-cross algorithm is applicable in the presented setting. Hence, the following description only applies to the special case of PSD matrices [KT92].

As an **iterative method**, the criss-cross algorithm goes through a sequence of basic solutions which are defined as follows:

Definition: Given a subset $B \subseteq \{1, \ldots, k\}$, $k = m + n$. The j-th column of the corresponding $k \times k$ matrix M_B is defined as equal to

$$\begin{array}{ll} I_j & \text{if } j \in B \text{ or} \\ -M_j & \text{if } j \notin B, \end{array} \tag{5}$$

where I_j is the j-th column of the $k \times k$ identity matrix I and $-M_j$ is the j-th column of the matrix $-M$. Note that B is called a **basis** if M_B is invertible.

For example, $B_{def} := \{1, ..., m + n\}$ is a basis because $M_B = I$ which is also the default start basis of the criss-cross algorithm.

Definition: Given a basis, the corresponding **basic solution** (w, z) is obtained as the unique solution of the following system of equations:

$$\begin{array}{rcll} z_j & = & 0 & \text{if } j \in B, \\ w_j & = & 0 & \text{if } j \notin B, \\ w - Mz & = & q. & \end{array} \tag{6}$$

The system (6) has a **unique solution** because the substitution of the first two sets of equations into $w - Mz = q$ results in

$$M_B \lambda_B = q,$$

where $(\lambda_B)_j = w_j$ if $j \in B$ and $(\lambda_B)_j = z_j$ otherwise.

As one can easily see, every basic solution (w, z) satisfies $w^T z = 0$, but $w, z \geq 0$ may not hold. The criss-cross algorithm tries to rectify this by **repeatedly moving** to another basis and the corresponding basic solution until $w, z \geq 0$ holds, in which case the LCP is solved.

Given a basis B along with λ_B^* which is the unique solution of $M_B \lambda_B = q$, **one step** of the criss-cross algorithm works as follows:

CRISS-CROSS ALGORITHM

1 | **IF** $\lambda_B^* \geq 0$ **THEN** stop because the vectors w and z satisfy the LCP [(4), p.8];

2 | **ELSE** choose the smallest index $r \in \{1, \ldots, m+n\}$ so that $(\lambda_B^*)_r < 0$ holds;

With respect to B, the system $w - Mz = q$ can be written as

$$M_B \lambda_B + M_N \lambda_N = q,$$

where $N = \{1, \ldots, m+n\} \setminus B$. Consequently,

$$\lambda_B = M_B^{-1} q - M_B^{-1} M_N \lambda_N$$

holds for all solutions of $w - Mz = q$. Let the $k \times k$ matrix $\Lambda = -M_B^{-1} M_N$ be the **dictionary** associated with B, so

$$\lambda_B = M_B^{-1} q + \Lambda \lambda_N. \tag{7}$$

There are now **two cases**:

2.1 | **IF** $\Lambda_{rr} > 0$ **THEN** update B to $B' := B \oplus \{r\}$ (**diagonal pivot**)[2];

2.2 | **ELSE IF** $\Lambda_{rr} = 0$ **THEN** choose the smallest index s so that $\Lambda_{rs} > 0$ and update B to $B' := B \oplus \{r, s\}$ (**exchange pivot**);

If no such index s exists, $\Lambda_{rj} \leq 0$ holds for all $j \in \{1, \ldots, m+n\}$. By (7),

$$(\lambda_B)_r = (\lambda_B^*)_r + \Lambda^r \lambda_N$$

holds for all solutions of $w - Mz = q$, where Λ^r is the r-th row of Λ. This results in $(\lambda_B)_r < 0$ whenever $\lambda_N \geq 0$. Hence, there cannot be any solution to $w - Mz = q$ with $w, z \geq 0$ and consequently the **LCP is unsolvable**.

2.3 | **ELSE** $\Lambda_{rr} < 0$ is impossible because M is a PSD matrix [KT92, Algorithm I].

It is well-known that the presented criss-cross algorithm **terminates** after having gone through a finite number of bases [KT92].

[2] $\oplus$ denotes the symmetric set difference.

4.1.4 Varying the Regularization Parameter

In the following, the variation of the **right-hand side** $q(\mu)$ of the pLCP [(4), p.8] is described. Assuming that the pLCP is solved for $\mu = \mu_{\min}$ using the criss-cross algorithm (4.1.3), a basis B [(5), p.9] has been found so that

$$\lambda_B^*(\mu) = M_B^{-1} q(\mu) \geq 0$$

holds [(6), p.9]. Given that b and c in the pQP [(2), p.7] are linear functions, $\lambda_B^*(\mu)$ and $q(\mu)$ depend linearly on μ. Hence, the **largest value** $\mu' := \mu + \hat{\mu} \geq \mu$ so that $\lambda_B^*(\mu') \geq 0$ holds can be easily computed, where $\mu' = \mu$ may hold as well as $\mu' = \infty$. Assuming that $q(\mu') = d\mu' + e$ holds for $(m+n)$-vectors d and e, it follows that

$$\begin{aligned} \lambda_B^*(\mu') &= M_B^{-1} q(\mu') \\ &= M_B^{-1}(d\hat{\mu} + d\mu + e) \\ &= \hat{\mu} M_B^{-1} d + M_B^{-1} q(\mu). \end{aligned} \tag{8}$$

Obviously, $\lambda_B^*(\mu')$ is the optimal solution for all $\mu' \geq \mu$ if $M_B^{-1} d \geq 0$. Otherwise, there exists at least one negative entry $\left(M_B^{-1} d\right)_r$, $r \in \{1, \ldots, m+n\}$. To compute the **largest valid** $\hat{\mu}$, the equation (8) is set equal to zero:

$$0 = \hat{\mu} M_B^{-1} d + \lambda_B^*(\mu).$$

Afterwards, the largest valid $\hat{\mu}$ is given by:

$$\hat{\mu} = \min \left\{ \frac{(-\lambda_B^*(\mu))_r}{\left(M_B^{-1} d\right)_r} \mid \left(M_B^{-1} d\right)_r < 0 \right\}. \tag{9}$$

With $\hat{\mu}$, B is valid throughout the **whole interval** $[\mu, \mu']$ and $\lambda_B^*(\kappa)$ is still a solution to the pLCP for every value $\kappa \in [\mu, \mu']$ and the right-hand side $q(\kappa)$.

In order to trace the **solution beyond** μ', the criss-cross algorithm is applied to the pLCP again. Now it starts with the basis B and the right-hand side $q = q(\mu' + \varepsilon)$, where ε is a symbolic parameter meant to represent an arbitrarily small positive value. Thus, the slightly perturbed pLCP is solved by starting from the solution to the old pLCP. In practice, the criss-cross method is expected to take only very few iterations this time although there are no theoretical guarantees for this: The complexity behavior is expected to be very similar to running Simplex steps for a slightly perturbed linear program, starting from a solution for the original problem.

While increasing μ, the interval $[\mu_{\min}, \mu_{\max}]$ is **subdivided** into pieces over which the solution to the pLCP and therefore also the solution to the pQP are linear in μ. There is only a finite number of such pieces because a basis B cannot occur twice: If B is valid for two values μ, μ', it is also valid for any intermediate value.

4.1.5 Odds and Ends

By using the analysis above, one can easily see that the criss-cross method computes the entire regularization path of any parametric quadratic program in **finite time**. When running the algorithm, the relevant size of the matrices M_B [(5), p.9] is bound by the number of non-zero entries in x plus m.

The regularization path computed in the way above may be **discontinuous** because the solution to the pLCP [(4), p.8] may "jump" when the parameter μ moves from $q(\mu')$ to $q(\mu' + \varepsilon)$. This is due to the facts that the pLCP generally does not have a unique solution and that the criss-cross method cannot control which optimal solution it finds. However, if continuity is strictly wanted, it is possible to insert connecting straight-line segments: Since both endpoints are solutions for $q(\mu')$, all intermediate points will be solutions as well. This holds for the x-part of (w, z) by convexity of the optimal region in the pQP [(2), p.7] and also for (w, z) with respect to the pLCP by a result of Adler and Gale [AG75].

Note that it is not necessary to assume that the pQP has an optimal solution throughout $[\mu_{\min}, \mu_{\max}]$. The criss-cross method can handle the **general case**: It may start at $\mu = \mu_{\min}$ with an unsolvable pLCP or it may run into an unsolvable situation later. Both will be reported by the second step of the algorithm if $\Lambda_{rr} = 0$ and $\Lambda_{rj} \leq 0$ hold for all $j \in \{1, \ldots, m+n\}$. In order to trace μ through such a situation, the method simply has to choose the "next event" as the largest $\mu' \geq \mu$ for which $(\lambda_B^*(\mu'))_r \geq 0$. Here, $(\lambda_B^*(\mu))_r$ is the variable for which infeasibility is detected during the criss-cross algorithm (4.1.3).

4.2 Support Vector Machines

Primarily, a support vector machine (SVM) is a **large margin classifier** to separate a set of objects into two classes so that the separative hyperplane has the largest possible distance to the nearest objects of any class. SVMs are used for regression as well as classification, see for example [SS02], [Abe05] or [Wan05]. Their origin lies in supervised machine learning and they were first mentioned by Vladimir Vapnik and Corinna Cortes in 1995, see [VC95]. For a condensed introduction see for example [Wan05, Support Vector Machines - An Introduction].

This **subsection** is structured as follows: First, some basic definitions concerning the problem setup are given. Then, both the primal hard and soft margin SVM are described. Afterwards, the Lagrange duality is presented and used on the primal soft margin SVM (SVM_{PSM}) to transform it into its dual equivalent. Finally, the benefit of using kernels together with dual soft margin SVMs (SVMs_{DSM}) is illustrated.

4.2.1 Basic Concepts

First, some definitions of basic concepts are stated. Except the training set definition, all of them are correspondingly[3] taken from [SS02]; the numbers in brackets refer to the page.

Definition (581): A set $\mathcal{H}$ is called a **vector space** (or linear space) over $\mathbb{R}$ if addition and scalar multiplication are defined, and satisfy (for all $x, x', x'' \in \mathcal{H}$, and $\eta, \eta' \in \mathbb{R}$)

$$
\begin{aligned}
&x + (x' + x'') = (x + x') + x'',\\
&x + x' = x' + x,\\
&x + 0 = x,\ 0 \in \mathcal{H},\\
&-x + x = 0,\ -x \in \mathcal{H},\\
&\eta x \in \mathcal{H},\\
&1x = x,\\
&\eta\,(\eta' x) = (\eta\eta')\,x,\\
&\eta\,(x + x') = \eta x + \eta x',\\
&(\eta + \eta')\,x = \eta x + \eta' x.
\end{aligned}
$$

The first four conditions amount to saying that $(\mathcal{H}, +)$ is a commutative group.

Definition: The set $\mathcal{T} := \left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{m}$, $x^{(i)} \in \mathcal{H}$, $y^{(i)} \in \{\pm 1\}$ is called the **training set**, where the $x^{(i)}$ are the independent variables (pattern vectors) and the $y^{(i)}$ are the dependent variables (labels). Additionally, the following is defined:

$$\mathcal{T}_{\pm} := \left\{\left(x^{(i)}, y^{(i)}\right) \in \mathcal{T} \mid \operatorname{sgn}\left(y^{(i)}\right) = \pm 1\right\}.$$

Definition (189): [...] given a dot product space $\mathcal{H}$, and a set of pattern vectors $x^{(1)}, \ldots, x^{(m)} \in \mathcal{H}$. Any **hyperplane** in $\mathcal{H}$ can be written as

$$\{x \in \mathcal{H} \mid w^T x + b = 0\},\ w \in \mathcal{H},\ b \in \mathbb{R}.$$

Since $\{x \in \mathcal{H} \mid w^T x + b = 0\} = \{x \in \mathcal{H} \mid c \cdot w^T x + c \cdot b = 0\}$ holds for any $c \in \mathbb{R},\ c \neq 0$, a hyperplane cannot be described uniquely. Therefore, w and b need to be scaled according to the training set $\mathcal{T}$:

Definition (190): The pair $(w, b) \in \mathcal{H} \times \mathbb{R}$ is called a **canonical hyperplane** with respect to $x^{(i)}, \ldots, x^{(m)} \in \mathcal{H}$, if it is scaled so that

$$\min_{i=1,\ldots,m} \left|w^T x^{(i)} + b\right| = 1,$$

which amounts to saying that the point closest to the hyperplane has a distance of $1/\|w\|$ [...].

[3] No changes with regard to content; only modifications in notation and highlighting.

Definition (192): For a hyperplane $\{x \in \mathcal{H} \mid w^T x + b = 0\}$, we call

$$\rho_{(w,b)}(x, y) := \frac{y\left(w^T x + b\right)}{\|w\|}$$

the **geometrical margin of the point** $(x, y) \in \mathcal{H} \times \{\pm 1\}$. The minimum value

$$\rho_{(w,b)} := \min_{i=1,\ldots,m} \rho_{(w,b)}\left(x^{(i)}, y^{(i)}\right)$$

shall be called the **geometrical margin** of $\left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)$. If the latter is omitted, it is understood that the training set is meant. Occasionally, we will omit the qualification geometrical, and simply refer to the **margin**.

As one can easily see, the absolute value of the margin of the point $(x, y) \in \mathcal{H} \times \{\pm 1\}$ is the **distance** from x to the hyperplane $\{x \in \mathcal{H} \mid w^T x + b = 0\}$. Remember the definition of the canonical hyperplane with $\rho_{(w,b)} = 1/\|w\|$.

Note that the n-dimensional **real coordinate space** $\mathbb{R}^n$ as a prototypical representative of a vector-space is used in all further explanations instead of $\mathcal{H}$.

4.2.2 Primal SVM

With the definitions in the previous subsection, all tools are on hand to define the well-known **primal SVM formalization**, see for example [SS02, p.12] or [Abe05, p.17],

$$\begin{array}{rl} \min_{w \in \mathbb{R}^n, b \in \mathbb{R}} & \frac{1}{2}\|w\|^2 \\ \text{s.t.} & y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 \ , \end{array} \tag{10}$$

where $i = 1, \ldots, m$ for given a training set $\mathcal{T}$ (4.2.1). Note that $\min \frac{1}{2}\|w\|^2$ is equal to **maximizing the margin** (4.2.1), $\max \frac{2}{\|w\|^2}$. Furthermore, the assumption is that there is at least one representative of each class given, $|\mathcal{T}_+|, |\mathcal{T}_-| \geq 1$.

The primal SVM formalization (10) has a solution if and only if the training set $\mathcal{T}$ is **linearly separable**. Otherwise, at least one $\left(x^{(i)}, y^{(i)}\right) \in \mathcal{T}$ must lead to $y^{(i)}\left(w^T x^{(i)} + b\right) < 0$ for which the constraint cannot be fulfilled.

In the case of linear separability, the constraint guarantees that a separative canonical hyperplane (4.2.1) is obtained. Hence, the **decision function** is given by

$$f_{w,b}(x) = \operatorname{sgn}\left(w^T x + b\right) = y, \tag{11}$$

where $x \in \mathbb{R}^n$ is unclassifiable if $f_{w,b}(x) = 0$ holds.

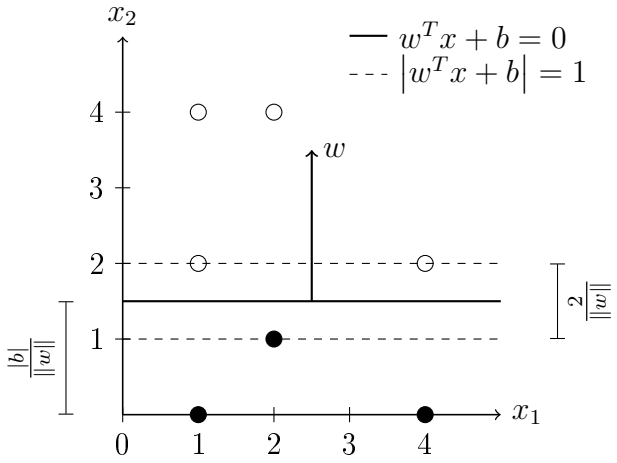

Figure 2: Simple SVM example (5.6.12) in which $w^T = (0,2)^T$ and $b = -3$ define the separating hyperplane of the two classes $\circ$ and $\bullet$.

Nevertheless, total linear separability is hard to guarantee; think for example of measurement errors or other outliers. Therefore, the **primal soft margin SVM** (SVM_{PSM}) formalization, see for example [SS02, p.16] or [Abe05, p.23],

$$\boxed{\begin{array}{lll} \textbf{SVM}_{\textbf{PSM}} & \min_{w\in\mathbb{R}^n, b\in\mathbb{R}} & \frac{1}{2}\|w\|^2 + c\cdot\sum_{i=1}^m \xi_i \\ & \text{s.t.} & y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 - \xi_i \\ & & \xi_i \geq 0 \end{array}} \tag{12}$$

is used in practice, where $i = 1, \ldots, m$ and slack-variables $\xi_i \geq 0$ are introduced to measure the degree of misclassification. Additionally, the parameter $c \in \mathbb{R}$ controls the **trade-off** between enlarging the margin and reducing the error. Nonetheless, linear separability is assumed in principle.

If linear separability of the training set $\mathcal{T}$ cannot be assumed, it is essential to have the possibility of determining a non-linear separation between the classes. The dual form of the SVM_{PSM} can easily handle this case by applying the so-called **kernel trick** (4.2.5). The Lagrange duality which is necessary to transform the SVM_{PSM} into its dual equivalent is described in the following.

4.2.3 Lagrange Duality

Given a primal optimization problem

$$\begin{array}{rl} \min_{x\in\mathbb{R}^n} & f(x) \\ \text{s.t.} & c_i(x) \leq 0\ , \end{array} \tag{13}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $c_i : \mathbb{R}^n \to \mathbb{R}$ are convex [(1), p.6] and continuously differentiable functions, the **KKT saddle point condition** [SS02, p.166] guarantees the following:

Theorem: Assume an optimization problem of the form (13) [...] and a Lagrangian

$$L(x,\alpha) := f(x) + \sum_{i=1}^{m} \alpha_i c_i(x) \text{ where } \alpha_i \geq 0. \tag{14}$$

If a pair of variables $(\bar{x}, \bar{\alpha})$ with $\bar{x} \in \mathbb{R}^n$ and $\bar{\alpha_i} \geq 0$ for all $i \in [m]$ exists, so that for all $x \in \mathbb{R}^n$ and $\alpha \in [0,\infty)^m$,

$$L(\bar{x},\alpha) \leq L(\bar{x},\bar{\alpha}) \leq L(x,\bar{\alpha}) \text{ (Saddle Point)} \tag{15}$$

then $\bar{x}$ is a solution to (13).

Note that the variables α_i are called **Lagrange multipliers** and that they become the dual variables in the SVM_{DSM} formalization [(21), p.17] in (4.2.4).

Proposition: The optimal value of the objective function is equal to the saddle point's value:

$$f(\bar{x}) = L(\bar{x},\bar{\alpha}). \tag{16}$$

Proof: Given a saddle point $(\bar{x},\bar{\alpha})$ with $\bar{x} \in \mathbb{R}^n$ and $\bar{\alpha} \geq 0$. Since $L(\bar{x},\alpha) \leq L(\bar{x},\bar{\alpha}) \leq L(x,\bar{\alpha})$ holds for all $x \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}_+^m$ according to (15), with (14), it follows that

$$f(\bar{x}) + \sum_{i=1}^{m} \alpha_i c_i(\bar{x}) \leq f(\bar{x}) + \sum_{i=1}^{m} \bar{\alpha_i} c_i(\bar{x}). \tag{17}$$

Assumption: There exists an index $j \in \{1,\ldots,m\}$ with $\bar{\alpha_j} c_j(\bar{x}) \neq 0$. It follows that

$$\bar{\alpha_j} c_j(\bar{x}) < 0$$

because $c_i(\bar{x}) \leq 0$ according to (13) and $\alpha_i \in \mathbb{R}_+$.

Choosing $\alpha = \bar{\alpha}$ except for $\alpha_j = \bar{\alpha_j} - \varepsilon > 0$ with $\varepsilon > 0$ leads to a conflict with (17) because

$$\alpha_j c_j(\bar{x}) = (\bar{\alpha_j} - \varepsilon) c_j(\bar{x}) > \bar{\alpha_j} c_j(\bar{x}),$$

which implies that $\bar{\alpha_j} c_j(\bar{x}) = 0$ holds for all $j \in \{1,\ldots,m\}$ and consequently (16) holds.

□

Corollary 1: According to (15) and (16), the dual optimization problem results in

$$\begin{array}{rl} \max_{\alpha\in\mathbb{R}^m} \min_{x\in\mathbb{R}^n} & L\left(x,\alpha\right) = f\left(\bar{x}\right) \\ \text{s.t.} & \alpha \geq 0\ . \end{array} \tag{18}$$

Corollary 2: Since f and c_i are convex and continuously differentiable functions (13), it follows for $\min_{x\in\mathbb{R}^n} L\left(x,\alpha\right)$ at $\bar{x}$ that

$$\frac{\partial}{\partial x} L\left(x,\alpha\right)_{|x=\bar{x}} = 0. \tag{19}$$

Corollary 3: According to (18) and (19), the final formalization of the dual optimization problem is given by:

$$\begin{array}{rl} \max_{\alpha\in\mathbb{R}^m} & L\left(x,\alpha\right) \\ \text{s.t.} & \frac{\partial}{\partial x} L\left(x,\alpha\right) = 0 \\ & \alpha \geq 0\ . \end{array} \tag{20}$$

4.2.4 Dual SVM

The derivation of the **dual soft margin SVM** (SVM_{DSM}) uses the Lagrange duality in analogy to the derivation of dual optimization problem formalization [(20), p.17]. Here, the SVM_{PSM} [(12, p.15)] is taken as the given optimization problem for the Lagrangian [(14), p.16].

Proposition: With the Lagrange dual optimization problem and the SVM_{PSM} as primal optimization problem, the SVM_{DSM} formalization results in

$$\boxed{\begin{array}{lrl} \textbf{SVM}_{\textbf{DSM}} & \max_{\alpha\in\mathbb{R}^m} & -\frac{1}{2}\sum_{i,j=1}^{m} \alpha_i\alpha_j y^{(i)}y^{(j)}x^{(i)^T}x^{(j)} + \sum_{i=1}^{m}\alpha_i \\ & \text{s.t.} & \sum_{i=1}^{m}\alpha_i y^{(i)} = 0 \\ & & 0 \leq \alpha \leq c \end{array}} \tag{21}$$

for a given training set $\mathcal{T}$ (4.2.1), the Lagrange multipliers α_i and $c \in \mathbb{R}$.

Proof: Consider the Lagrangian for the given SVM_{PSM},

$$L\left(w,b,\xi,\alpha,\beta\right) = \frac{1}{2}\left\|w\right\|^2 + c\cdot\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\alpha_i\left(y^{(i)}\left(w^T x^{(i)} + b\right) - 1 + \xi_i\right) - \beta^T\xi, \tag{22}$$

where $\alpha, \beta \in \mathbb{R}_+^m$. Corresponding to the Lagrange dual optimization problem formalization, the partial derivatives of the Lagrangian are:

$$\frac{\partial}{\partial w} L\left(w, b, \xi, \alpha, \beta\right) = w - \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} = 0 \tag{23}$$

$$\frac{\partial}{\partial b} L\left(w, b, \xi, \alpha, \beta\right) = \sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

$$\frac{\partial}{\partial \xi_i} L\left(w, b, \xi, \alpha, \beta\right) = c - \alpha_i - \beta_i = 0$$

Plugging these into (22) results in

$$\begin{array}{rl} \max_{\alpha,\beta \in \mathbb{R}^m} & -\frac{1}{2} \sum_{i,j}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)^T} x^{(j)} + \sum_{i,j}^{m} \alpha_i \\ \text{s.t.} & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \\ & \alpha_i + \beta_i = c, \ i = 1, \ldots, m \\ & \alpha, \beta \geq 0 \ . \end{array}$$

With $\Big[(\alpha_i, \beta_i \geq 0) \wedge (\alpha_i + \beta_i = c)\Big] \rightarrow \big[\alpha_i \leq c\big]$ finally follows (21).

□

In analogy to [(11), p.14], the **decision function** obtained after solving the SVM_{DSM} is given by

$$f_{\alpha,b}\left(x\right) = \text{sgn}\left(\sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)^T} x + b\right) = y, \tag{24}$$

where $w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$ holds according to (23) and $x \in \mathbb{R}^n$ is unclassifiable if $f_{\alpha,b}\left(x\right) = 0$ holds. The problem of determining b is addressed in [(29), p.21].

Remember the need for determining a non-linear separation between the classes if linear separability of the training set $\mathcal{T}$ cannot be assumed (4.2.2). By using the SVM_{DSM} formalization, the **kernel trick** can be applied easily to obtain a non-linear separation which is described in the following subsection.

4.2.5 Kernels

The basic idea behind kernels is to map the pattern vectors x of the training set $\mathcal{T}$ (4.2.1) into a higher-dimensional **feature space**,

$$x \in \mathbb{R}^n \rightarrow \Phi\left(x\right) \in \mathbb{R}^f, \ n, f \in \mathbb{N}, \ n \leq f,$$

and to solve the classification problem there. The assumption is that the SVM [(12), p.15 & (21), p.17] is able to **separate the images** of x linearly in the higher-dimensional $\mathbb{R}^f$.

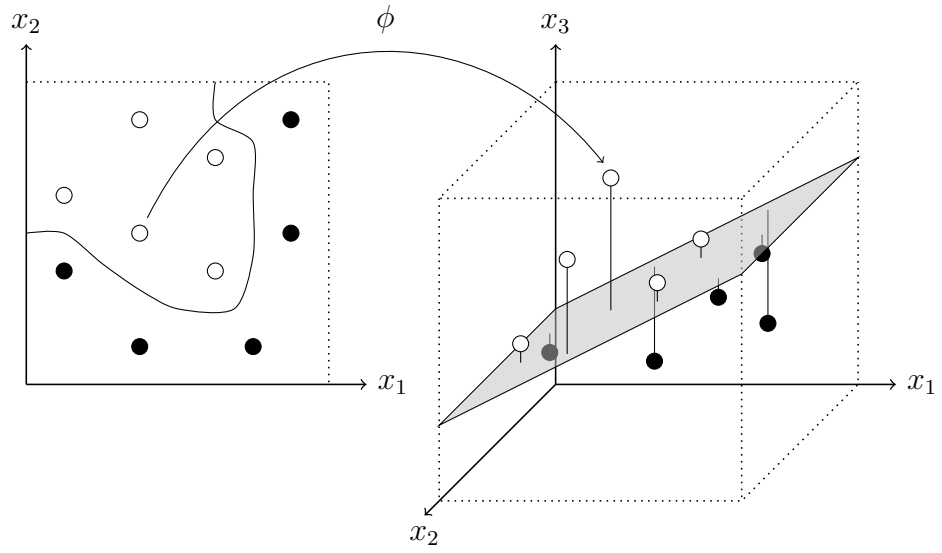

Figure 3: Illustration of mapping two linearly non-separable classes ∘ and • into a higher-dimensional feature space where they become linearly separable.

First, kernels are formally defined. The following definitions are correspondingly[4] taken from [Abe05, subsection 2.3]; the numbers in brackets refer to the page.

Definition (313): Let $K(x, x')$ be a real-valued symmetric function with x and x' being n-dimensional vectors. For any set of data $\{x^{(1)}, \ldots, x^{(m)}\}$ and $h = (h_1, \ldots, h_m)^T$ with m being any natural number, if

$$h^T K h \geq 0$$

is satisfied (i.e., K is a positive semidefinite matrix), we call $K(x, x')$ a **positive semidefinite kernel**, where

$$K = \begin{pmatrix} K\left(x^{(1)}, x^{(1)}\right) & \cdots & K\left(x^{(1)}, x^{(m)}\right) \\ \vdots & \ddots & \vdots \\ K\left(x^{(m)}, x^{(1)}\right) & \cdots & K\left(x^{(m)}, x^{(m)}\right) \end{pmatrix}.$$

Definition (25): According to the **Hilbert-Schmidt theory** [...], if a symmetric function $K(x, x')$ satisfies

$$\sum_{i,j=1}^{m} h_i h_j K(x_i, x_j) \geq 0 \tag{25}$$

[4]No changes with regard to content; only modifications in notation and highlighting.

for all m, x_i and h_i, where m takes on a natural number and h_i take on real numbers, there exists a **mapping function**, $\Phi(x)$, that maps x into the dot-product feature space and $\Phi(x)$ satisfies

$$K(x, x') = \Phi^T(x)\,\Phi(x').$$

[...] The condition (25) [...] is called **Mercer's condition**, and the function that satisfies (25) [...] is called the [...] Mercer kernel. In the following, if there is no confusion, we simply call it the kernel.

Definition (26): The advantage of using kernels is that we need not treat the high-dimensional feature space explicitly. This technique is called **kernel trick**. Namely, we use $K(x, x')$ in training and classification instead of $\Phi(x)$.

The use of $\Phi(x)$ would be necessary if the idea of using kernels were to be applied to the $\mathbf{SVM_{PSM}}$ [(12), p.15] because the pattern vectors x of the training set $\mathcal{T}$ appear only in the constraint. Therefore, the constraint is the only place where the transformation into the higher-dimensional feature space can take place:

$$\begin{array}{rl} \min_{w\in\mathbb{R}^f, b\in\mathbb{R}} & \frac{1}{2}\|w\|^2 + c\cdot\sum_{i=1}^m \xi_i \\ \text{s.t.} & y^{(i)}\left(w^T\Phi\left(x^{(i)}\right)+b\right) \geq 1-\xi_i,\ i=1,\ldots,m \\ & \xi_i \geq 0\ . \end{array}$$

In contrast to this, the $\mathbf{SVM_{DSM}}$ formalization [(21), p.17] allows to take advantage of the kernel trick: As $x^{(i)^T}x^{(j)}$ appears in the objective function, it can be replaced by any positive semidefinite kernel $\Phi^T\left(x^{(i)}\right)\Phi\left(x^{(j)}\right) = K\left(x^{(i)}, x^{(j)}\right)$.

The following positive semidefinite kernels are **mainly used** in SVMs, see for example [SS02, subsection 2.3] or [Abe05, subsection 2.3.2]:

Linear Kernel: $K_l(x, x') = x^T x'$

Polynomial Kernel: $K_p(x, x') = \left(x^T x' + a\right)^d$, $d \in \mathbb{N}$, $a \geq 0$

Gaussian Kernel: $K_g(x, x') = \exp\left(-\gamma\|x - x'\|^2\right)$, $\gamma \geq 0$

Sigmoid Kernel: $K_s(x, x') = \tanh\left(\kappa\cdot x^T x' + a\right)$, $\kappa > 0$, $a < 0$

In analogy to [(24), p.18], the **decision function** obtained after solving the $\mathrm{SVM_{DSM}}$ using any positive semidefinite kernel $K(x, x')$ is given by

$$f_{\alpha,b}(x) = \operatorname{sgn}\left(\sum_{i=1}^m \alpha_i y^{(i)} K\left(x^{(i)}, x\right) + b\right) = y, \tag{26}$$

where $x \in \mathbb{R}^n$ is unclassifiable if $f_{\alpha,b}(x) = 0$ holds.

The value of b is determined using the following **KKT complementarity conditions** [Abe05, p.26], where $i = 1, \ldots, m$:

$$\alpha_i \left(y^{(i)} \left(\sum_{j=1}^{m} \alpha_j y^{(j)} K\left(x^{(i)}, x^{(j)}\right) + b \right) - 1 + \xi_i \right) = 0 \tag{27}$$

$$(c - \alpha_i)\, \xi_i = 0 \tag{28}$$

$$\alpha_i, \xi_i \geq 0$$

Proposition: Given one unbound support vector[5] and its label $\left(\tilde{x}^{(i)}, \tilde{y}^{(i)}\right) \in \mathcal{T}$, b is determined by

$$b = \tilde{y}^{(i)} - \sum_{j=1}^{m} \alpha_j y^{(j)} K\left(\tilde{x}^{(i)}, x^{(j)}\right). \tag{29}$$

Proof: Since $0 < \alpha_i$ holds for all unbound support vectors $\tilde{x}^{(i)}$, the following equation needs to be equal to zero to satisfy (27):

$$\tilde{y}^{(i)} \left(\sum_{j=1}^{m} \alpha_j y^{(j)} K\left(\tilde{x}^{(i)}, x^{(j)}\right) + b \right) - 1 + \xi_i = 0.$$

This leads to

$$b = \tilde{y}^{(i)} - \sum_{j=1}^{m} \alpha_j y^{(j)} K\left(\tilde{x}^{(i)}, x^{(j)}\right) - \frac{\xi_i}{\tilde{y}^{(i)}},$$

given that $\tilde{y}^{(i)} \in \{\pm 1\}$. As $\alpha_i < c$ holds for all unbound support vectors $\tilde{x}^{(i)}$ as well, $\xi_i = 0$ must hold to fulfill (28). This finally results in (29).

□

For **numerical stability**, the determination of b should involve all unbound support vectors and their labels by computing the arithmetic average as follows

$$b = \frac{1}{|\mathcal{U}|} \sum_{\left(\tilde{x}^{(i)}, \tilde{y}^{(i)}\right) \in \mathcal{U}} \left(\tilde{y}^{(i)} - \sum_{j=1}^{m} \alpha_j y^{(j)} K\left(\tilde{x}^{(i)}, x^{(j)}\right) \right),$$

where $\mathcal{U} := \left\{ \left(\tilde{x}^{(i)}, \tilde{y}^{(i)}\right) \mid 0 < \alpha_i < c \right\} \subseteq \mathcal{T}$ is the set of all unbound support vectors.

For **additional information** about kernels, see for example [SS02, chapter 2] or [Abe05, subsection 2.3].

[5] Pattern vectors $\tilde{x}^{(i)}$ of the training set $\mathcal{T}$ are called unbound support vectors if $0 < \alpha_i < c$ holds.

4.3 SVM to pQP

When looking at the SVM_{DSM} formalization [(21), p.17], one can easily see that it does not fit the pQP formalization [(2), p.7]. Since the latter is a precondition for the criss-cross method (4.1), a **transformation** of the SVM_{DSM} into a pQP is necessary.

Proposition: The following transformation into a pQP still fits the SVM_{DSM} formalization:

$$\begin{array}{rl} \min_{\alpha\in\mathbb{R}^m} & \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y^{(i)}y^{(j)}x^{(i)^T}x^{(j)} - \sum_{i=1}^{m}\alpha_i \\ \text{s.t.} & \sum_{i=1}^{m}\alpha_i y^{(i)} \geq 0 \\ & -\sum_{i=1}^{m}\alpha_i y^{(i)} \geq 0 \\ & \alpha \geq 0 \\ & -\alpha \geq -c\ . \end{array} \tag{30}$$

Proof 1: First, it has to be shown that the transformation above is indeed a pQP. Therefore, $Q_{ij} = y^{(i)}y^{(j)}x^{(i)^T}x^{(j)}$ must be a PSD matrix. Remember that the SVM_{DSM} uses kernels[6] which are positive semidefinite (4.2.5). Hence, the following equation,

$$\alpha^T Q\alpha = \tilde{\alpha}^T K\tilde{\alpha} \geq 0,$$

holds for any $\tilde{\alpha}$, where $\tilde{\alpha}_i = \alpha_i \cdot y_i$, $i = 1, \ldots, m$ and so Q is a PSD matrix.

□

Proof 2: As one can easily see, the following propositions,

- $\left[\max_{\alpha\in\mathbb{R}^m} -\frac{1}{2}\alpha^T Q\alpha + \sum_{i=1}^{m}\alpha_i\right] \leftrightarrow \left[\min_{\alpha\in\mathbb{R}^m} \alpha^T Q\alpha - \sum_{i=1}^{m}\alpha_i\right]$,
- $\left[\sum_{i=1}^{m}\alpha_i y^{(i)} = 0\right] \leftrightarrow \left[\left(-\sum_{i=1}^{m}\alpha_i y^{(i)} \geq 0\right) \wedge \left(\sum_{i=1}^{m}\alpha_i y^{(i)} \geq 0\right)\right]$ and
- $\left[0 \leq \alpha \leq c\right] \leftrightarrow \left[(\alpha \geq 0) \wedge (-\alpha \geq -c)\right]$,

are true, where $Q_{ij} = y^{(i)}y^{(j)}x^{(i)^T}x^{(j)}$.

□

Corollary: According to (30), the resulting pQP which can be transformed into a pLCP [(4), p.8] later is given by

[6] In the default formalization, an SVM_{DSM} uses the linear kernel $K_l = (x, x') = x^T x'$, $x \in \mathbb{R}^n$.

$$\begin{array}{rl} \min_{\alpha\in\mathbb{R}^m} & \frac{1}{2}\alpha^T Q\alpha + \hat{c}^T\alpha \\ \text{s.t.} & A\alpha \geq b\,(c) \\ & \alpha \geq 0\ , \end{array} \tag{31}$$

where $Q_{ij} = y^{(i)}y^{(j)}x^{(i)^T}x^{(j)}$, $\hat{c}^T = (-1\ \ldots\ -1)$, $b^T = \left(0\ 0\ \underbrace{-c\ldots-c}_{m}\right)$ and

$$A = \begin{pmatrix} y^{(1)} & \cdots & y^{(m)} \\ -y^{(1)} & \cdots & -y^{(m)} \\ -1 & & 0 \\ & \ddots & \\ 0 & & -1 \end{pmatrix}. \tag{32}$$

In contrast to the original pQP, the parameter μ resides only in b represented by c.

With the transformation (31), the criss-cross method can be applied to SVMs_{DSM}. Hence, the **kernel trick** (4.2.5) which is used to create a non-linear separation between the classes is compatible with the criss-cross method.

Another advantage of the transformation (31) is the decrease of the **combinatorial complexity** in contrast to the primal form. The latter is hard to convert to a pQP because the constraints $w, b \geq 0$ are missing in the SVM_{PSM} formalization [(12), p.15] which would be necessary for an easy transformation into a pQP.

Proposition: The following transformation into a pQP,

$$\begin{array}{rl} \min_{\hat{x}\in\mathbb{R}^{2(n+1)+m}} & \frac{1}{2}\hat{x}^T Q\hat{x} + \hat{c}\,(c)^T\,\hat{x} \\ \text{s.t.} & A\hat{x} \geq \hat{b} \\ & \hat{x} \geq 0\ , \end{array} \tag{33}$$

still fits the SVM_{PSM} formalization [(12), p.15], where $\hat{x}^T = \left(w_+^T, w_-^T, b_+, b_-, \xi^T\right)$ with $w = w_+ - w_-$ and $b = b_+ - b_-$,

$$Q = \begin{pmatrix} I & -I & \\ -I & I & \\ & & 0 \\ & & \end{pmatrix}$$

with I being the $n \times n$ identity matrix, $\hat{c}^T = \left(\underbrace{0\ldots0}_{2(n+1)}\ \underbrace{c\ldots c}_{m}\right)$, $\hat{b}^T = (1\ldots1)$ and

$$A = \begin{pmatrix} y^{(1)}x^{(1)^T} & -y^{(1)}x^{(1)^T} & y^{(1)} & -y^{(1)} & 1 & & \\ \vdots & \vdots & \vdots & \vdots & & \ddots & \\ y^{(m)}x^{(m)^T} & -y^{(m)}x^{(m)^T} & y^{(m)} & -y^{(m)} & & & 1 \end{pmatrix}.$$

Proof 1: First, it has to be shown that the transformation above is indeed a pQP. Therefore, the matrix Q defined above must be a PSD matrix. This is the case, because the following equation,

$$\hat{x}^T Q \hat{x} = w_+^T w_+ - 2w_+^T w_- + w_-^T w_- = \|w_+ - w_-\|^2 \geq 0$$

holds, where $\hat{x}^T = \left(w_+^T, w_-^T, b_+, b_-, \xi^T\right)$.

Proof 2: As one can easily see, the following propositions,

- $\left[w \in \mathbb{R}^n\right] \leftrightarrow \Big[(w = w_+ - w_-) \wedge (w_+, w_- \geq 0)\Big]$ and
- $\left[b \in \mathbb{R}\right] \leftrightarrow \Big[(b = b_+ - b_-) \wedge (b_+, b_- \geq 0)\Big]$,

are true and the following equations,

- $\hat{x}^T Q \hat{x} = w_+^T w_+ - 2w_+^T w_- + w_-^T w_- = \|w_+ - w_-\|^2 = \|w\|^2$ and
- $\left(y^{(i)}x^{(i)^T} w_+ - y^{(i)}x^{(i)^T} w_-\right) + \left(y^{(i)} b_+ - y^{(i)} b_-\right) + \xi_i = y^{(i)}\left(x^{(i)^T} w + b\right) + \xi_i$,

hold, where $w = w_+ - w_-$, $b = b_+ - b_-$ and $i = 1, \ldots, m$.

□

Remark: In contrast to the original pQP, the parameter μ resides only in $\hat{c}$ represented by c.

With respect to (33), the matrix M in the pLCP is a $2\,(m+n+1) \times 2\,(m+n+1)$ matrix, whereas M is a $2\,(m+1) \times 2\,(m+1)$ matrix for (31). Since the criss-cross method is a **combinatorial type** algorithm, the dimension of M plays a critical role for the expected running time.

Another benefit can be obtained from the SVM_{DSM} formalization if $|\mathcal{T}_+| \approx |\mathcal{T}_-|$ holds for a given training set $\mathcal{T}$ (4.2.1). It appears that this combination results in a good time saving **start basis** B for the criss-cross method at the beginning of the regularization path compared to the default basis $B_{def} = \{1, \ldots, m+n\}$ (4.1.3). This circumstance is dealt with in (4.4.3).

For all these reasons, the SVM_{DSM} transformation into a pQP according to (30) and (31) is **highly recommended** if the criss-cross method is to be applied to soft margin SVMs.

4.4 Basis Reconstruction

In practice, the criss-cross method (4.1) needs most of its running time to determine the **first basic solution** [(6), p.9] of a given pLCP [(4), p.8] where $\lambda_B^* \geq 0$ holds when using the default basis $B_{def} = \{1, \ldots, m+n\}$ [Wel10, p.30].

Opposite to that, the computation of the **regularization path** is relatively fast. Note that a growing parameter μ narrows the width of the margin (4.2.1). Therefore, pattern vectors move from being inside the margin to the outside while residing on the margin in between (4.6). Since every constellation is represented by a unique basis B (4.1.4), the running time of the entire path computation depends linearly on the number of pattern vectors. For experimental results see (5.6).

Since the interest lies in the **regularization path** only, it is important to determine the basis B [(5), p.9] resulting in $\lambda_B^*(\mu) \geq 0$ for a fixed parameter μ as fast as possible to save most of the running time.

In the following, a **distinction** is drawn between arbitrary pQPs[7] [(2), p.7] in (4.4.1) and SVM_{DSM}-pQPs [(31), p.23] with $|\mathcal{T}_+| = |\mathcal{T}_-|$ (4.2.1) in (4.4.3). An important difference between these two cases is that the general case can use an arbitrary parameter μ, whereas the special case has to start with the parameter $c = 0$ because it is associated with the start of the regularization path.

4.4.1 General Case

Remember the system of equations used to obtain the unique **basic solution** [(6), p.9] for a given basis B. There

$$\begin{array}{rcl} z_j & = & 0,\ j \in B \text{ and} \\ w_j & = & 0,\ j \notin B \end{array} \tag{34}$$

have to hold, where $w^T = \left(u^T, v^T\right)$ and $z^T = \left(x^T, y^T\right)$ according to the pLCP formalization [(4), p.8]. As one can easily see, the following is equivalent to (34):

$$\begin{array}{rcl} \big[j \in B\big] & \rightarrow & \big[z_j = 0\big], \\ \big[j \notin B\big] & \rightarrow & \big[w_j = 0\big]. \end{array}$$

To reconstruct the basis B, both z and w must be known because, for example, $z_j = 0$ may hold for any $j \notin B$ as well. To decide whether $j \in B$ or not, the following equivalences must be true:

$$\begin{array}{rcl} \big[j \in B\big] & \leftrightarrow & \Big[(z_j = 0) \wedge (w_j \neq 0)\Big], \\ \big[j \notin B\big] & \leftrightarrow & \Big[(w_j = 0) \wedge (z_j \neq 0)\Big]. \end{array} \tag{35}$$

[7] Note that SVM_{DSM}-pQPs [(31), p.23] with arbitrary $\mathcal{T}$ (4.2.1) are handled in this case as well.

For the two missing logical propositions,

$$\begin{array}{lcl} (z_j \neq 0) & \wedge & (w_j \neq 0) \text{ and} \\ (z_j = 0) & \wedge & (w_j = 0) \ , \end{array} \tag{36}$$

the first can never be true because (34) holds for any basic solution for a given basis B, whereas the latter is true for all remaining indices $\hat{j} \in B$ that do not fit any equivalence in (35).

In consequence, **two problems** have to be solved successively to reconstruct the basis: Determine the vectors z and w of a given pLCP and decide for all remaining $\hat{j}$ whether $\hat{j} \in B$ or not.

Determining the vectors z and w: According to the **KKT optimality conditions** [(3), p.8] $w^T = \left(u^T, v^T\right)$ and $z^T = \left(x^T, y^T\right)$ appear in

$$\begin{array}{lll} \text{(i)} & v = Ax - b\left(\mu\right) \geq 0 & \text{and} \quad x \geq 0 \\ \text{(ii)} & u = c\left(\mu\right) - A^T y + Qx \geq 0 & \text{and} \quad y \geq 0 \ , \end{array}$$

where (i) encodes the primal feasibility of x and (ii) encodes the dual feasibility of y. Therefore, x is the solution of the given primal pQP [(2), p.7] and y is the solution of its dual equivalent. The dual pQP can be obtained by using the **Lagrange duality** (4.2.3) with the primal pQP as the given optimization problem [(13), p.15] for the Lagrangian [(14), p.16].

Proposition: With the Lagrange dual optimization problem formalization [(20), p.17] and the pQP as primal optimization problem, the dual pQP formalization results in

$$\begin{array}{rl} \max_{\alpha \in \mathbb{R}^m} & -\frac{1}{2} x^T Q x + y^T b\left(\mu\right) \\ \text{s.t.} & A^T y \leq Q^T x + c\left(\mu\right) \\ & y \geq 0 \ , \end{array} \tag{37}$$

with an $n \times n$ symmetric PSD matrix Q, an $m \times n$ matrix A, linear functions $c : \mathbb{R} \to \mathbb{R}^n$ and $b : \mathbb{R} \to \mathbb{R}^m$ as well as the solution x of the primal pQP.

Proof: Consider the Lagrangian for the primal pQP:

$$L\left(x, y, z\right) = \frac{1}{2} x^T Q x + c\left(\mu\right)^T x - y^T \left(Ax - b\left(\mu\right)\right) - z^T x \tag{38}$$

where $y, z \in \mathbb{R}^m_+$. Corresponding to the Lagrange dual optimization problem formalization, the partial derivative of the Lagrangian is:

$$\frac{\partial}{\partial x} L\left(x, y, z\right) = x^T Q + c\left(\mu\right)^T - y^T A - z^T = 0$$

Plugging this into (38) results in

$$\begin{array}{rl} \max_{y,z\in\mathbb{R}^m} & -\frac{1}{2}x^TQx + y^Tb\,(\mu) \\ \text{s.t.} & y^TA + z^T = x^TQ + c\,(\mu)^T \\ & y, z \geq 0\ . \end{array}$$

With $\Big[(y_i, z_i \geq 0) \wedge \Big(y^TA + z^T = x^TQ + c\,(\mu)^T\Big)\Big] \rightarrow \Big[A^Ty \leq Q^Tx + c\,(\mu)\Big]$ finally follows (37).

□

Note that the constraints in (37) correspond to the **KKT optimality conditions'** second condition which encodes dual feasibility of y because Q is symmetric.

According to proposition (37), the solution x of the primal pQP is necessary to generate its dual equivalent. Since the criss-cross method is to be avoided for computing x as mentioned in the beginning of this subsection, **other software** has to be used.

We use the proprietary software **IBM ILOG CPLEX**[8] in version 12.1.0 to solve the SVM_{DSM}-pQP [(31), p.23] and the corresponding dual pQP (37), both for a fixed parameter c respectively μ. Afterwards, the equations in the KKT optimality conditions must be solved to determine the vectors u and v. Finally, with $w^T = \big(u^T, v^T\big)$ and $z^T = \big(x^T, y^T\big)$ known, the basis can be reconstructed using (35) except for the undetermined indices $\hat{j}$.

Note that the **accuracy** (5.3.3) decreases with a growing parameter μ due to the restricted machine accuracy while checking z and w against zero in (35) and (36) to reconstruct the basis. Furthermore, there is additional effort necessary to determine a "good" parameter μ in order to be able to compute the entire regularization path, see (4.4.2). Remember that the parameter μ resides only in b represented by c for an SVM_{DSM}-pQP.

Handling the indices $\hat{j}$: Since $z_{\hat{j}}, w_{\hat{j}} = 0$ for all indices $\hat{j}$, no decision on whether $\hat{j} \in B$ or not can be made with the help of (35). Therefore, **all possibilities** have to be checked by testing all matrices $M_{b\in\hat{B}}$ [(5), p.9] successively as long as one is invertible with

$$\hat{B} := \Big\{B \cup P \mid P \in \wp\Big(\hat{S}\Big)\Big\} \text{ and } B = \big\{j \mid (z_j = 0) \wedge (w_j \neq 0)\big\}, \tag{39}$$

where $\wp\Big(\hat{S}\Big)$ is the power set of $\hat{S} := \big\{\hat{j} \mid \big(z_{\hat{j}} = 0\big) \wedge \big(w_{\hat{j}} = 0\big)\big\}$. As w and z satisfy the pLCP, **at least one** $b' \in \hat{B}$ must define an invertible matrix $M_{b'}$.

[8] There is also a free trail edition available which allows "time- and problem size-limited use of all of the product's features for evaluation purposes" according to the manufacturer's website.

Once an invertible matrix $M_{b'}$ is found, $\lambda^*_{b'} \geq 0$ in step 1 of the criss-cross algorithm (4.1.3) may not hold for the given parameter μ because more than one $b' \in \hat{B}$ could lead to an invertible matrix due to the indeterminable indices $\hat{j}$. Therefore, **additional rounds** of the criss-cross algorithm are necessary to determine the basis b^* defining the optimal solution where $\lambda^*_{b^*} \geq 0$ holds. Nevertheless, the basis b' is close to b^* because at least B (39) is determined correctly.

An **alternative** could be to search for M_{b^*} among all invertible matrices $M_{b'}$ without using the criss-cross method.

4.4.2 Determining the Path's Start

According to (4.4.1), additional effort is necessary to determine a "good" c to be able to compute the **entire regularization path** for a given SVM_{DSM}-pQP [(31), p.23] with $|\mathcal{T}_+| \neq |\mathcal{T}_-|$ (4.2.1). Otherwise, $c = 0$ is used (4.4.4) in the special case where $|\mathcal{T}_+| = |\mathcal{T}_-|$ holds (4.4.3).

We used the free software **LIBSVM** in version 2.91 [CCCJ01] to estimate c_0 so that the computation of the regularization path starts at its beginning. The software solves SVMs_{DSM} [(21), p.17] for a given parameter c and supports all types of kernels mentioned in (4.2.5). Moreover, LIBSVM is able to compute the prediction accuracy (40). The software is freely available as library source code in C++ and Java and as binary executable for Microsoft Windows.

To determine c_0, **different small values** c'_i have to be tested for both the training and a prediction set. To continue, the following definition has to be stated:

Definition: Given a prediction set $\mathcal{P}$ defined in analogy to a training set $\mathcal{T}$. The **prediction accuracy** $\text{pa}(c)$ is defined as

$$\text{pa}(c) := \frac{\left|\left\{i \mid f_{\alpha(c),b(c)}\left(x^{(i)}\right) = y^{(i)}\right\}\right|}{|\mathcal{P}|}, \tag{40}$$

being the ration between correctly classified $\left(x^{(i)}, y^{(i)}\right) \in \mathcal{P}$ and $|\mathcal{P}|$ using the decision function $f_{\alpha,b}$ [(26), p.20] for a given parameter c.

Our recommendation is to test $c'_i = 10^{-i}$, $i = 0, \ldots, \infty$. At the point where $\text{pa}(c'_i)$ (40) **drops significantly** and remains the same for further c'_j, $j > i$, the value for $c_0 = c'_i$ is found. The reason for this characteristic is that the margin's width (4.2.1) is maximal for c_0 (4.6), where as many as possible pattern vectors $x^{(i)}$ of the training set, $i \in \{1, \ldots, m\}$ lie inside the margin with their associated $\alpha_i = c_0$ (4.4.4).

To validate c_0, the results of the criss-cross method (4.1) concerning the following c_j's and $\text{pa}(c_j)$'s can be used. Table 1 shows a **correctly determined** c_0 where wide gaps occur

between $j = 0$ and $j = 1$ for the values of c_j and $\mathrm{pa}\,(c_j)$ relative to the gaps between $j = 1$ and $j = 2$. Note that j indicates the j-th step along the regularization path (4.1.4)

j	c_j	$\mathrm{pa}\,(c_j)$
0	0.00100	64.7059
1	0.00125	75.2451
2	0.00127	75.7353

Table 1: Correctly determined parameter c_0.

In contrast, table 2 shows a c_0 which is **chosen incorrectly** because of the unfulfilled reasons mentioned above.

j	c_j	$\mathrm{pa}\,(c_j)$
0	0.01000	80.3922
1	0.01001	80.3922
2	0.01011	80.3922

Table 2: Incorrectly determined parameter c_0.

4.4.3 Special Case

In contrast to the preceding subsection, the interest lies now in $\mathrm{SVM_{DSM}}$-pQPs [(31), p.23] where $|\mathcal{T}_+| = |\mathcal{T}_-|$ (4.2.1) holds.

Definition: B_0^* is the start basis of an $\mathrm{SVM_{DSM}}$-pQP if it defines the matrix $M_{B_0^*}$ [(5), p.9] for which $\lambda_{B_0^*}^* \geq 0$ holds in step 1 of the criss-cross algorithm (4.1.3) at the start of the regularization path for a matching parameter c.

For any start basis B_0^* at the beginning of the regularization path, it appears in our experiments that $|B_0^*| = 2$ holds. In the following, the structure of this **artificial basis** B_{art} is defined. As B_{art} is the start basis B_0^* of any $\mathrm{SVM_{DSM}}$-pQP where $|\mathcal{T}_+| = |\mathcal{T}_-|$ holds, the criss-cross method (4.1) can be applied using B_{art} to compute the regularization path directly. A proof for this statement is given in (4.4.4).

Proposition: Given the artificial start basis $B_{art} := \{m+1, m+2\}$. The following matrix $M_{B_{art}}$ based on an $\mathrm{SVM_{DSM}}$-pQP,

$$M_{B_{art}} = \begin{pmatrix} -Q & A'^T \\ -A & 0' \end{pmatrix}, \tag{41}$$

is invertible, where the $m \times (m+2)$ matrix A'^T and the $(m+2) \times (m+2)$ matrix $0'$ are given by

$$A'^T := \begin{pmatrix} 0 & 0 & -1 & & 0 \\ \vdots & \vdots & & \ddots & \\ 0 & 0 & 0 & & -1 \end{pmatrix}, \; 0' := \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & & \\ & & 0 & \end{pmatrix}.$$

Proof: The matrix $M_{B_{art}}$ is invertible if and only if all column vectors are linearly independent. As one can easily see, the column vectors of A [(32), p.23] are linearly independent which consequently holds for all columns in

$$\begin{pmatrix} -Q \\ -A \end{pmatrix}$$

as well. In analogy, all column vectors of A'^T are linearly independent except for the first two. For these, the matrix $0'$ supports the linear independence of

$$\begin{pmatrix} A'^T \\ 0' \end{pmatrix}$$

with its first two linearly independent columns. Since all parts in the above matrices that guarantee linear independence for themselves cannot affect each other, linear independence follows for (41).

□

Consider the case when no external software should be used to determine the start basis B_0^* for a **general $\mathbf{SVM_{DSM}}$-pQP** where $|\mathcal{T}_+| = |\mathcal{T}_-|$ does not have to hold. In this case, it may be better to use the artificial basis B_{art} instead of the default basis $B_{def} = \{1, \ldots, 2m+2\}$ (4.1.3) when starting the criss-cross algorithm. Since $|B_0^*| = 2\,|(|\mathcal{T}_+| - |\mathcal{T}_-|)| + 2$ may be small as well, B_{art} could be closer to the start basis B_0^* relative to the default basis B_{def}. For a detailed description see (4.4.4).

Be further $|\mathcal{T}_+|, |\mathcal{T}_-|$ so that $|B_0^*| - |B_{art}| \approx |B_{def}| - |B_0^*|$ holds. To decide whether to use the default basis B_{def} or the artificial basis B_{art} to compute the start basis B_0^* using the criss-cross method, the **expected running time** is of the utmost importance.

One benefit of using B_{def} affects the **LU decomposition** (5.2.1) of the matrix M_B used in the criss-cross method. As columns of $-M$ enter the matrix M_B during the criss-cross algorithm (4.1.3) in steps 2.1 and 2.2, the running time of the LU decomposition increases depending on the density of M_B. However, its speed is as fast as possible at the start of the criss-cross method because $M_{B_{def}} = I$. This also holds for the matrix $M_{B_{art}}$ because the matrix $M_{B_{art}}$ can be transformed into an upper triangular matrix as well. This transformation is a permutation σ of rows and columns in the original matrix

$$M_{B_{art}} = \begin{pmatrix} & & & 0 & 0 & -1 & & \\ & -Q & & \vdots & \vdots & & \ddots & \\ & & & 0 & 0 & & & -1 \\ \pm 1 & \dots & \pm 1 & 1 & 0 & 0 & \dots & 0 \\ \mp 1 & \dots & \mp 1 & 0 & 1 & 0 & \dots & 0 \\ 1 & & & 0 & 0 & & & \\ & \ddots & & \vdots & \vdots & & 0 & \\ & & 1 & 0 & 0 & & & \end{pmatrix}$$

which results in

$$M^{\sigma}_{B_{art}} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & \pm 1 & \dots & \pm 1 \\ 0 & 1 & 0 & \dots & 0 & \mp 1 & \dots & \mp 1 \\ 0 & 0 & -1 & & & & & \\ \vdots & \vdots & & \ddots & & & -Q & \\ 0 & 0 & & & -1 & & & \\ 0 & 0 & & & & 1 & & \\ \vdots & \vdots & & 0 & & & \ddots & \\ 0 & 0 & & & & & & 1 \end{pmatrix}.$$

Therefore, nearly no additional **time expenditure** is necessary except for the permutation σ during the LU decomposition. Note that this holds only when starting the criss-cross algorithm to determine the start basis B_0^* with the artificial basis B_{art}.

Nevertheless, the question of **practical usability** remains: If $|\mathcal{T}_{\pm}| >> |\mathcal{T}_{\mp}|$ holds, the only possibility to generate $|\mathcal{T}_+| \approx |\mathcal{T}_-|$ is to locate and delete unnecessary representatives of the predominant class in the training set $\mathcal{T}$. This process is very hard to verify because deep knowledge about the training set is necessary.

Subsection (4.5) will give a short introduction to **conjoint analysis** where $|\mathcal{T}_+| \approx |\mathcal{T}_-|$ can easily be generated by doubling the number of constraints for the SVM_{PSM} [(10), p.14].

4.4.4 Artificial Basis Derivation

As mentioned in (4.4.3), the artificial basis $B_{art} = \{m+1, m+2\}$ is the start basis B_0^* for the criss-cross method (4.1) applied on an SVM_{DSM}-pQP [(31), p.23] if $|\mathcal{T}_+| = |\mathcal{T}_-|$ (4.2.1) holds. This subsection shows why the basis B_{art} is defined as it is.

Proposition: Given a training set $\mathcal{T}$ with $|\mathcal{T}_+| = |\mathcal{T}_-|$. The optimal solution of an SVM_{DSM} [(21), p.17] is given by $\alpha = c$ for any sufficiently small parameter $c > 0$.

Proof: Remember the objective function of the $\mathrm{SVM}_{\mathrm{DSM}}$,

$$\max_{\alpha \in \mathbb{R}^m} -\frac{1}{2} \sum_{i,j \in \mathcal{M}} \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)^T} x^{(j)} + \sum_{i \in \mathcal{M}} \alpha_i, \tag{42}$$

where $\mathcal{M} := \{1, \ldots, m\}$.

It has to be shown that (42) is maximal for $\alpha = c$ for any sufficiently small parameter $c > 0$. Note that the constraints of the $\mathrm{SVM}_{\mathrm{DSM}}$ hold because of $|\mathcal{T}_+| = |\mathcal{T}_-|$ and $\alpha = c$.

The following transformation,

$$-\frac{1}{2} \left(\sum_{i,j \in \mathcal{M} \setminus \{\hat{j}\}} \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)^T} x^{(j)} + 2 \sum_{i \in \mathcal{M}} \alpha_i \alpha_{\hat{j}} y^{(i)} y^{(\hat{j})} x^{(i)^T} x^{(\hat{j})} \right) + \sum_{i \in \mathcal{M}} \alpha_i, \tag{43}$$

is equivalent to (42), where $\hat{j} \in \mathcal{M}$ is an arbitrarily chosen index. With $\alpha = c$ according to the assumption and $t = \sum_{i,j \in \mathcal{M} \setminus \{\hat{j}\}} c^2 y^{(i)} y^{(j)} x^{(i)^T} x^{(j)}$, it follows that (43) is equal to

$$-\frac{1}{2} \left(t + 2 \sum_{i \in \mathcal{M}} c^2 y^{(i)} y^{(\hat{j})} x^{(i)^T} x^{(\hat{j})} \right) + \sum_{i \in \mathcal{M} \setminus \{\hat{j}\}} c + c. \tag{44}$$

Be further $\alpha_{\hat{j}} = c - \varepsilon \geq 0$, $0 < \varepsilon \leq c$. With (43), it follows that (43) is equal to

$$-\frac{1}{2} \left(t + 2 \sum_{i \in \mathcal{M}} \alpha_i (c - \varepsilon) y^{(i)} y^{(\hat{j})} x^{(i)^T} x^{(\hat{j})} \right) + \sum_{i \in \mathcal{M} \setminus \{\hat{j}\}} c + (c - \varepsilon). \tag{45}$$

Now it has to be shown that (44) $\geq$ (45) for any sufficiently small parameter $c > 0$:

$$\left[-\sum_{i \in \mathcal{M}} c^2 y^{(i)} y^{(\hat{j})} x^{(i)^T} x^{(\hat{j})} + c \right] \overset{!}{\geq} \left[-\sum_{i \in \mathcal{M}} \alpha_i (c - \varepsilon) y^{(i)} y^{(\hat{j})} x^{(i)^T} x^{(\hat{j})} + (c - \varepsilon) \right]. \tag{46}$$

Therefore, the following inequation has to hold:

$$(c - \varepsilon) - c \leq \left[\sum_{i \in \mathcal{M}} \alpha_i (c - \varepsilon) y^{(i)} y^{(\hat{j})} x^{(i)^T} x^{(\hat{j})} - \sum_{i \in \mathcal{M}} c^2 y^{(i)} y^{(\hat{j})} x^{(i)^T} x^{(\hat{j})} \right].$$

Note that

$$\sum_{i \in \mathcal{M}} \alpha_i \alpha_{\hat{j}} y^{(i)} y^{(\hat{j})} x^{(i)^T} x^{(\hat{j})} = (|\mathcal{M}| - 1)\, \alpha_{\hat{j}} \sum_{i \in \mathcal{M} \setminus \{\hat{j}\}} \alpha_i y^{(i)} y^{(\hat{j})} x^{(i)^T} x^{(\hat{j})} + \alpha_{\hat{j}}^2 y^{(\hat{j})^2} x^{(\hat{j})^T} x^{(\hat{j})}$$

holds. With

$$o = (|\mathcal{M}| - 1) \cdot \textstyle\sum_{i \in \mathcal{M} \setminus \{\hat{j}\}} y^{(i)} y^{(\hat{j})} x^{(i)^T} x^{(\hat{j})} \quad \text{and} \quad p = y^{(\hat{j})^2} x^{(\hat{j})^T} x^{(\hat{j})} \geq 0$$

and therefore

$$\begin{array}{rcl} \sum_{i \in \mathcal{M}} \alpha_i (c - \varepsilon) y^{(i)} y^{(\hat{j})} x^{(i)^T} x^{(\hat{j})} & = & c(c - \varepsilon) \cdot o + (c - \varepsilon)^2 \cdot p, \\ \sum_{i \in \mathcal{M}} c^2 y^{(i)} y^{(\hat{j})} x^{(i)^T} x^{(\hat{j})} & = & c^2 \cdot o + c^2 \cdot p \end{array}$$

it follows for (46) that

$$\begin{array}{rcl} -\varepsilon & \leq & \big(c(c - \varepsilon) \cdot o + (c - \varepsilon)^2 \cdot p\big) - \big(c^2 \cdot o + c^2 \cdot p\big) \\ & \leq & \big(c(c - \varepsilon) - c^2\big) \cdot o + \big((c - \varepsilon)^2 - c^2\big) \cdot p \\ & \leq & \big((c - \varepsilon) - c\big) \cdot c \cdot o + \big((c - \varepsilon) - c\big)\big((c - \varepsilon) + c\big) \cdot p \\ & \leq & -\varepsilon\big(c \cdot o + (2c - \varepsilon) \cdot p\big) \\ 1 & \geq & c \cdot (o + 2p) - \varepsilon \cdot p \end{array}$$

because $0 < \varepsilon \leq c$. Thus, the proposition is true for any

$$0 \leq c \leq \frac{1 + \varepsilon \cdot p}{o + 2p}$$

if $o + 2p \neq 0$ because the inequations hold for $c = \varepsilon = 0$ at least. Otherwise, $c > 0$ can be chosen arbitrarily because $1 \geq -\varepsilon \cdot p$ holds for any c since $p \geq 0$.

□

Remark: Remember that $|\mathcal{T}_+| = |\mathcal{T}_-|$ holds according to the assumption. Be $\left(x^{(\hat{j})}, y^{(\hat{j})}\right) \in \mathcal{T}_-$ w.l.o.g. Since the constraint $\sum_{i=1}^m \alpha_i y^{(i)} = 0$ has to hold, a $\left(x^{(\hat{\hat{j}})}, y^{(\hat{\hat{j}})}\right) \in \mathcal{T}_+$ must exist with $\alpha_{\hat{\hat{j}}} = \alpha_{\hat{j}}$. Nevertheless, the correctness of the proof above is not affected because it holds with $\alpha_{\hat{j}}$ replaced by $\alpha_{\hat{\hat{j}}}$ too.

Corollary 1: If $o + 2p = 0$ holds, the objective function is maximal for $\alpha = c$ and any $c \geq 0$. Therefore, there is no regularization path to compute.

Be $o + 2p \neq 0$ in the following.

Corollary 2: The structure of the start basis B_0^* is given by the proposition above. According to it, the objective function is maximal for $\alpha = c$,

$$c \in \left[0, \frac{1 + \varepsilon \cdot p}{o + 2p}\right].$$

Referring to the system of equations [(6), p.9] used to obtain the unique basic solution for a given basis B, $i \notin B_0^*$ if $\alpha_i = c$ holds because $z^T = \left(\alpha^T, y^T\right)$. Additionally, $v_i = 0$, $i = 3, \ldots, m$ because $v_i = -\alpha_i + c$ holds according to the KKT optimality conditions [(3), p.8]. Since $w^T = \left(u^T, v^T\right)$, the basis B_0^* does not need to contain the indices $i = m + 3, \ldots, 2m + 2$ either.

At last, to avoid the matrix $M_{B_0^*}$ from not being invertible, both $i = m + 1$ and $i = m + 2$ must be elements of B_0^* as described in (4.4.3). Consequently and under the consideration of (4.6), the start basis results in $B_0^* = B_{art}\{m + 1, m + 2\}$.

Remark: As the objective function is maximal for any

$$c \in \left[0, \frac{1 + \varepsilon \cdot p}{o + 2p}\right],$$

the criss-cross method can be applied to an SVM_{DSM}-pQP with the parameter $c = 0$ and the start basis $B_0^* = B_{art}$ to compute the entire regularization path.

Now, consider a **general SVM_{DSM}-pQP** where $|\mathcal{T}_+| = |\mathcal{T}_-|$ does not have to hold. As mentioned in (4.4.3), the artificial basis B_{art} may be closer to the start basis B_0^* at the beginning of the regularization path relative to the default basis B_{def} (4.1.3) because $|B_0^*| = 2\,|(|\mathcal{T}_+| - |\mathcal{T}_-|)| + 2$ holds. The following proposition is used to prove this circumstance.

Proposition: Given a training set $\mathcal{T}$ with $|\mathcal{T}_+| = l$, $|\mathcal{T}_-| = k$ and $l > k$ w.l.o.g. The optimal solution of an SVM_{DSM} is given by

- $\alpha_{i_-} = c$, $i_- \in \left\{i \mid \left(x^{(i)}, y^{(i)}\right) \in \mathcal{T}_-\right\}$, $\mathcal{C}_- = \{i_-\}$,
- $\alpha_{i_+} = c$, $i_+ \in \left\{i \mid \left(x^{(i)}, y^{(i)}\right) \in \mathcal{T}_+\right\}$, $\mathcal{C}_+ = \{i_+\}$ and
- $\alpha_{j_+} = 0$, $j_+ \in \left\{j \mid \left(x^{(j)}, y^{(j)}\right) \in \mathcal{T}_+\right\} \setminus \mathcal{C}_+$, $\mathcal{O}_+ = \{j_+\}$

for a sufficiently small parameter $c > 0$, where $|\mathcal{C}_-| = |\mathcal{C}_+| = k$ must hold to satisfy the constraint $\sum_{i=1}^{l+k} \alpha_i y^{(i)} = 0$ and $|\mathcal{O}_+| = l - k$.

Proof by Cases: It has to be shown that (42) is maximal for the variables chosen in the proposition.

1. Be $\alpha_{\hat{j}} = c - \varepsilon$, $\hat{j} \in \mathcal{C}_-$ arbitrarily chosen and $0 < \varepsilon \leq c$. An $\alpha_{\hat{\hat{j}}} = \alpha_{\hat{j}}$ with $\hat{\hat{j}} \in \mathcal{C}_+$ has to be chosen because the constraint $\sum_{i=1}^{l+k} \alpha_i y^{(i)} = 0$ has to hold. This decreases the objective function as shown in the first proof and remark above.
2. Be $\alpha_{\hat{j}} = c - \varepsilon$, $\hat{j} \in \mathcal{C}_+$ arbitrarily chosen and $0 < \varepsilon \leq c$. There are two possibilities to satisfy the constraint $\sum_{i=1}^{l+k} \alpha_i y^{(i)} = 0$:
   (a) If an $\alpha_{\hat{\hat{j}}} = \alpha_{\hat{j}}$ with $\hat{\hat{j}} \in \mathcal{C}_-$ is chosen, the situation is in analogy to the first case.
   (b) If an $\alpha_{\hat{\hat{j}}} = c - \alpha_{\hat{j}} = \varepsilon$ with $\hat{\hat{j}} \in \mathcal{O}_+$ is chosen and $\mathcal{M} := \mathcal{C}_- \cup \mathcal{C}_+$, it suffices to show that

$$\left[- \sum_{i,j \in \mathcal{M}} c^2 y^{(i)} y^{(j)} x^{(i)^T} x^{(j)} \right] \overset{!}{\geq} \left[- \sum_{i,j \in \mathcal{M} \cup \left\{\hat{\hat{j}}\right\}} \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)^T} x^{(j)} \right] \tag{47}$$

holds for the value of the objective function and any sufficiently small parameter $c > 0$ because

$$\sum_{i \in \mathcal{M}} c = \sum_{i \in \mathcal{M} \setminus \left\{\hat{j}\right\}} c + \alpha_{\hat{j}} + \alpha_{\hat{\hat{j}}}$$

holds. The inequation (47) leads to

$$\left[\sum_{i \in \mathcal{M}} c^2 y^{(i)} y^{(\hat{j})} x^{(i)^T} x^{(\hat{j})} \right] \leq \left[\sum_{i \in \mathcal{M}} \alpha_i (c - \varepsilon) y^{(i)} y^{(\hat{j})} x^{(i)^T} x^{(\hat{j})} + q \right] \tag{48}$$

similar to (46), where

$$q = \sum_{i \in \mathcal{M} \cup \left\{\hat{\hat{j}}\right\}} \alpha_i \alpha_{\hat{\hat{j}}} y^{(i)} y^{(\hat{\hat{j}})} x^{(i)^T} x^{(\hat{j})}.$$

With

$$o = (|\mathcal{M}| - 1) \cdot \textstyle\sum_{i \in \mathcal{M} \setminus \left\{\hat{j}\right\}} y^{(i)} y^{(\hat{j})} x^{(i)^T} x^{(\hat{j})} \quad \text{and} \quad p = y^{(\hat{j})^2} x^{(\hat{j})^T} x^{(\hat{j})} \geq 0,$$

the inequation (48) can be transformed into

$$\begin{aligned} c^2 \cdot o + c^2 \cdot p &\leq c(c-\varepsilon) \cdot o + (c-\varepsilon)^2 \cdot p + q \\ 0 &\leq -\varepsilon \left(c \cdot o + (2c - \varepsilon) \cdot p \right) + q \ . \end{aligned}$$

With

$$q = c\varepsilon \underbrace{\left(|\mathcal{M}| - 1 \right) \sum_{i \in \mathcal{M} \setminus \{\hat{j}\}} y^{(i)} y^{(\hat{j})} x^{(i)^T} x^{(\hat{j})} + \varepsilon(c-\varepsilon) x^{(\hat{j})^T} x^{(\hat{j})} + \varepsilon^2 x^{(\hat{j})^T} x^{(\hat{j})}}_{=:\hat{\sum}}$$

follows that

$$\begin{aligned} 0 &\geq c \cdot o + (2c - \varepsilon) \cdot p - c\hat{\sum} - (c-\varepsilon) x^{(\hat{j})^T} x^{(\hat{j})} - \varepsilon x^{(\hat{j})^T} x^{(\hat{j})} \\ &\geq c \cdot \left(o + 2p - \hat{\sum} - x^{(\hat{j})^T} x^{(\hat{j})} \right) - \varepsilon \cdot \left(p - x^{(\hat{j})^T} x^{(\hat{j})} + x^{(\hat{j})^T} x^{(\hat{j})} \right) \end{aligned}$$

because $0 < \varepsilon \leq c$. Finally, the proposition is true for any

$$0 \leq c \leq \frac{\varepsilon \cdot \left(x^{(\hat{j})^T} x^{(\hat{j})} - x^{(\hat{j})^T} x^{(\hat{j})} + x^{(\hat{j})^T} x^{(\hat{j})} \right)}{o + 2p - \hat{\sum} - x^{(\hat{j})^T} x^{(\hat{j})}}$$

if $\left(o + 2p - \hat{\sum} - x^{(\hat{j})^T} x^{(\hat{j})} \right) \neq 0$ because the inequations hold trivially for $c = \varepsilon = 0$ at least. Otherwise, $c > 0$ can be chosen arbitrarily because $0 \geq -\varepsilon \cdot \left(x^{(\hat{j})^T} x^{(\hat{j})} - x^{(\hat{j})^T} x^{(\hat{j})} + x^{(\hat{j})^T} x^{(\hat{j})} \right)$ holds for any c.

□

Corollary 1: If $\left(o + 2p - \hat{\sum} - x^{(\hat{j})^T} x^{(\hat{j})} \right) = 0$ holds, the objective function is maximal for $\alpha_{i_\pm} = c$, where $i_\pm \in \mathcal{C}_- \cup \mathcal{C}_+$ and any $c \geq 0$. Thus, there is no regularization path to compute.

Be $\left(o + 2p - \hat{\sum} - x^{(\hat{j})^T} x^{(\hat{j})} \right) \neq 0$ in the following.

Corollary 2: The structure of the start basis B_0^* is given by the proposition above. According to it, the objective function is maximal for $\alpha_{i_\pm} = c$ and $\alpha_{j_+} = 0$, where $i_\pm \in \mathcal{C}_- \cup \mathcal{C}_+$, $j_+ \in \mathcal{O}_+$ and

$$c \in \left[0, \frac{\varepsilon \cdot \left(x^{(\hat{j})^T} x^{(\hat{j})} - x^{(\hat{j})^T} x^{(\hat{j})} + x^{(\hat{j})^T} x^{(\hat{j})} \right)}{o + 2p - \hat{\sum} - x^{(\hat{j})^T} x^{(\hat{j})}} \right] .$$

Referring to the system of equations [(6), p.9] used to obtain the unique basic solution for a given basis B, $i_\pm \notin B_0^*$ if $\alpha_{i_\pm} = c$ holds because $z^T = \left(\alpha^T, y^T\right)$. Additionally, $v_{i_\pm+2} = 0$ because $v_{i_\pm+2} = -\alpha_{i_\pm} + c$ holds according to the KKT optimality conditions [(3), p.8]. Since $w^T = \left(u^T, v^T\right)$, the basis B_0^* does not need to contain the indices $j = l + k + i_\pm + 2$ either.

The same observations follow for the indices j_+, just the opposite way: According to the KKT optimality conditions, $v_{j_+ +2} = c$ because $v_{j_+ +2} = -\alpha_{j_+} + c$ holds. Hence, the basis B_0^* contains the indices $j = l + k + j_+ + 2$ because of [(6), p.9]. Additionally, the basis B_0^* can contain the indices j_+ because $\alpha_{j_+} = 0$ holds.

At last, to avoid the matrix $M_{B_0^*}$ from not being invertible, both $i = l + k + 1$ and $i = l + k + 2$ must be elements of B_0^* as described in (4.4.3). Consequently and under the consideration of (4.6), $|B_0^*| = 2(l - k) + 2$ must hold.

Remark: According to experimental results (5.6), $|B_0^*| \approx 2\,|l - k| + 2$ holds for all tested SVM_{DSM}-pQPs with $|\mathcal{T}_+| \neq |\mathcal{T}_-|$. There, the start bases B_0^* at the beginning of the regularization path are determined as described in (4.4.1). Thus, the search for an invertible matrix $M_{B_0^*}$ among all possible bases, with respect to the indeterminable indices $\hat{j}$ and the restricted machine accuracy, leads to the above approximation.

4.5 Conjoint Analysis

4.5.1 Introduction

CONsidered JOINTly analysis is part of the multivariate analytical methods and originated in mathematical psychology. The approach is to measure the research participants' preferences on options of artificial or real objects. These options are described by n attributes and their $|A_i|$, $i = 1, \ldots, n$ levels. The objective is to **determine the effect** of each attribute's level for the decision process. Today, conjoint analysis is mostly used in market research. There is a large number a related literature, see for example [BEPW08, chapter 9] or [BB09].

4.5.2 Problem Setup

Given a structured set of options $A = A_1 \times \ldots \times A_n$ with **well-ordered sets** A_i and measurements $\left(x^{(i)}, y^{(i)}\right)$, $i = 1, \ldots, m$, with

$$\begin{aligned} x^{(i)} &= \left(a^{(i)}, \bar{a}^{(i)}\right),\ a^{(i)}, \bar{a}^{(i)} \in A \\ y^{(i)} &= \begin{cases} +1, & a^{(i)} \succcurlyeq \bar{a}^{(i)} \\ -1, & a^{(i)} \preccurlyeq \bar{a}^{(i)} \end{cases}, \end{aligned} \tag{49}$$

where $a \succcurlyeq \bar{a}$ means that a **is preferred to** $\bar{a}$. To describe this quantitatively, the assumption is that the following proposition is true:

$$\left[a \succcurlyeq \bar{a}\right] \leftrightarrow \left[\hat{v}\left(a\right) > \hat{v}\left(\bar{a}\right)\right]. \tag{50}$$

The function $v_i : A_i \to \mathbb{R}$ describes the **part worth value** of $a_i \in A_i$. Thereby, the part worth value's vector $\hat{v}$ is defined by

$$\hat{v} := \sum_{i=1}^{n} v_i\left(a_i\right).$$

The assumption (50) implies that

$$\sum_{i=1}^{n} v_i\left(a_i\right) - v_i\left(\bar{a}_i\right) \geq \varepsilon,\ \varepsilon > 0$$

and $\hat{v}\left(a^{(i)}\right) \neq \hat{v}\left(\bar{a}^{(i)}\right)$, $i = 1, \ldots, m$. A **characteristic vector** $\chi_a \in \{0,1\}^{\hat{n}}$, $\hat{n} := \sum_{i=1}^{n} |A_i|$, can be defined for any $a \in A$ as

$$\chi_a := \left(\underbrace{0 \ldots 0 \overbrace{1}^{\mathrm{Ord}(a_1)} 0 \ldots 0}_{|A_1| \text{ entries}} \underbrace{0 \ldots 0 \overbrace{1}^{\mathrm{Ord}(a_2)} 0 \ldots 0}_{|A_2| \text{ entries}} \ldots \underbrace{0 \ldots 0 \overbrace{1}^{\mathrm{Ord}(a_n)} 0 \ldots 0}_{|A_n| \text{ entries}} \right)^T,$$

where $\mathrm{Ord}\left(a_i\right)$, $i = 1, \ldots, n$, indicates the **position** of a_i in the well-ordered set A_i. Be $v \in \mathbb{R}^{\hat{n}}$ the vector of all part worth values $v_i\left(a_i\right)$, $a_i \in A_i$ and $i = 1, \ldots, n$. Thus,

$$\sum_{i=1}^{n} \left(v_i\left(a_i\right) - v_i\left(\bar{a}_i\right)\right) - \varepsilon = n_{a\bar{a}}^T v - \varepsilon \geq 0 \tag{51}$$

holds with $n_{a\bar{a}} := \chi_a - \chi_{\bar{a}}$.

The **learning problem** is to determine v and ε. In geometrical terms, (51) means that v must linger in the positive open half-space whose limiting hyperplane (4.2.1) has the normal $n_{a\bar{a}}$. This implies that valid vectors v can only be found in the intersection of all closed half-spaces,

$$H_{a\bar{a}}^{+} := \left\{v \in \mathbb{R}^{\hat{n}} \mid n_{a\bar{a}}^T v - \varepsilon \geq 0\right\},\ H_{a\bar{a}}^{-} := \left\{v \in \mathbb{R}^{\hat{n}} \mid n_{a\bar{a}}^T v + \varepsilon \leq 0\right\},$$

defined by the measurements $\left(x^{(i)}, y^{(i)}\right)$, $i = 1, \ldots, m$.

Since $\bar{H}_{a\bar{a}}^{+} = \bar{H}_{\bar{a}a}^{-}$ holds, any **order** of a and $\bar{a}$ can be fixed. Furthermore, note that y is associated with $H_{a\bar{a}}^{\pm}$ in the following way:

$$\begin{array}{rcl} \left[y = +1\right] & \leftrightarrow & \bar{H}_{a\bar{a}}^{+} \\ \left[y = -1\right] & \leftrightarrow & \bar{H}_{a\bar{a}}^{-}\ . \end{array}$$

4.5.3 SVMs and Conjoint Analysis

It can be shown[9] that the problem of estimating v and ε can be transformed into the following $\mathbf{SVM_{PSM}}$,

$$\begin{array}{rl} \min_{v\in\mathbb{R}^{\hat{n}},b\in\mathbb{R}} & \frac{1}{2}\left\|v\right\|^2 + c\cdot\sum_{i=1}^{m}\xi_i \\ \text{s.t.} & y^{(i)}\left(v^T n_{a^{(i)}\bar{a}^{(i)}} - \varepsilon\right) \geq 1-\xi^{(i)} \\ & \xi^{(i)} \geq 0\ , \end{array} \tag{52}$$

with help of the **duality in projective geometry**. Note that the original SVM_{PSM} formalization [(12), p.15] is obtained by simply setting $\varepsilon = -b$.

As $\bar{H}^{+}_{a\bar{a}} = \bar{H}^{-}_{\bar{a}a}$ holds, the generation of $|\mathcal{T}_+| = |\mathcal{T}_-|$ (4.2.1) can be achieved by adding all vectors $n_{\bar{a}^{(i)}a^{(i)}}$ to the constraints of (52) as **redundant information**. Therefore, the extended SVM_{PSM} is given by

$$\begin{array}{rl} \min_{v\in\mathbb{R}^{\hat{n}},b\in\mathbb{R}} & \frac{1}{2}\left\|v\right\|^2 + 2c\cdot\sum_{i=1}^{m}\xi_i \\ \text{s.t.} & y^{(i)}\left(v^T n_{a^{(i)}\bar{a}^{(i)}} + b\right) \geq 1-\xi^{(i)} \\ & -y^{(i)}\left(v^T n_{\bar{a}^{(i)}a^{(i)}} - b\right) \geq 1-\xi^{(i)} \\ & \xi^{(i)} \geq 0\ . \end{array}$$

Since $+b = -b$ holds for the **optimal solution** of any dual soft margin SVM, b must be equal to zero. For experimental results see (5.6.6) and (5.6.7). Hence, the final SVM_{PSM} is given by

$$\begin{array}{rl} \min_{v\in\mathbb{R}^{\hat{n}},b\in\mathbb{R}} & \frac{1}{2}\left\|v\right\|^2 + c\cdot\sum_{i=1}^{2m}\xi_i \\ \text{s.t.} & \tilde{y}^{(i)}\left(v^T \tilde{n}_{a^{(i)}\bar{a}^{(i)}}\right) \geq 1-\xi^{(i)} \\ & \xi^{(i)} \geq 0 \end{array} \tag{53}$$

with $\left(\tilde{x}^{(i)},\tilde{y}^{(i)}\right)$ being defined for $i = 1,\ldots,m$ like (49) and for $i = m+1\ldots 2m$ as follows:

$$\begin{array}{rcl} \tilde{x}^{(i)} & = & \left(\bar{a}^{(i-m)}, a^{(i-m)}\right) \\ \tilde{y}^{(i)} & = & -y^{(i-m)}\ . \end{array}$$

Both formalizations (52) and (53) should be transformed into their dual equivalents (4.2.4) using the Lagrange duality (4.2.3) because of the reasons mentioned in (4.3) if the criss-cross method (4.1) is to be applied to an SVM_{PSM}. The $\mathbf{SVMs_{DSM}}$ for the problems (52) and (53) are given by

$$\begin{array}{rl} \max_{\alpha\in\mathbb{R}^m} & -\frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y^{(i)}y^{(j)} n^T_{a^{(i)}\bar{a}^{(i)}} n_{a^{(j)}\bar{a}^{(j)}} + \sum_{i=1}^{m}\alpha_i \\ \text{s.t.} & \sum_{i=1}^{m}\alpha_i y^{(i)} = 0 \\ & 0 \leq \alpha \leq c \end{array} \tag{54}$$

and

[9] A proof of this theory is the topic of future research under the leadership of Prof. Dr. Joachim Giesen at the University of Jena.

$$\begin{array}{rl} \max_{\alpha\in\mathbb{R}^{2m}} & -\frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j\tilde{y}^{(i)}\tilde{y}^{(j)}\tilde{n}^T_{a^{(i)}\bar{a}^{(i)}}\tilde{n}_{a^{(j)}\bar{a}^{(j)}}+\sum_{i=1}^{2m}\alpha_i \\ \text{s.t.} & \sum_{i=1}^{2m}\alpha_i\tilde{y}^{(i)}=0 \\ & 0\leq\alpha\leq c\ . \end{array} \tag{55}$$

To apply the criss-cross method to (54) and (55), both problems can be transformed into pQPs (4.3). Thereby, the **artificial basis** B_{art} (4.4.3) can be used with (55) to start the computation directly at the beginning of the regularization path (4.1.4).

4.5.4 Part Worth Values' Computation

Remember that the learning problem (4.5.2) is to determine the vector $v\in\mathbb{R}^{\hat{n}}$, $\hat{n}:=\sum_{i=1}^{n}|A_i|$ of all part worth values $v_i\,(a_i)$ beside ε. After applying the criss-cross method (4.1) to the **$\mathbf{SVM_{DSM}}$ formalization** of a conjoint analysis problem [(54), p.39 & (55), p.40], the vector v is computed from the solution vector α as follows:

$$v=\sum_{i=1}^{m}\alpha_i y^{(i)}n_{a^{(i)}\bar{a}^{(i)}}\ \text{resp.}\ v=\sum_{i=1}^{2m}\alpha_i\tilde{y}^{(i)}n_{a^{(i)}\bar{a}^{(i)}}.$$

These equations follow [(23), p.18] which is a result of applying the **Lagrange duality** (4.2.3) to transform an $\mathrm{SVM_{PSM}}$ [(12), p.15] into its dual equivalent.

Note that the part worth values $v\in\mathbb{R}^{\hat{n}}$ cannot be computed if a **non-linear kernel** (4.2.5) is used because the pattern vectors $n_{a^{(i)}\bar{a}^{(i)}}$ are mapped into a higher-dimensional feature space:

$$n_{a^{(i)}\bar{a}^{(i)}}\in\mathbb{R}^{\hat{n}}\rightarrow\Phi\left(n_{a^{(i)}\bar{a}^{(i)}}\right)\in\mathbb{R}^{f},\ f\in\mathbb{N},\ \hat{n}<f.$$

Therefore, the resulting vector

$$v'=\sum_{i=1}^{m}\alpha_i y^{(i)}\Phi\left(n_{a^{(i)}\bar{a}^{(i)}}\right)$$

also resides in the $\mathbb{R}^f$ where v' does not represent the original part worth values v any more.

4.6 Understanding the Geometry

Remember the **KKT complementarity conditions** used in [(27), p.21] to determine the value of b,

$$\alpha_i\left(y^{(i)}f\left(x^{(i)}\right)-1+\xi_i\right)=0 \tag{56}$$

$$(c-\alpha_i)\,\xi_i=0 \tag{57}$$

$$\alpha_i, \xi_i \geq 0,$$

where $i = 1, \ldots, m$ and

$$f\left(x^{(i)}\right) = \sum_{j=1}^{m} \alpha_j y^{(j)} K\left(x^{(i)}, x^{(j)}\right) + b = w^T \Phi\left(x^{(i)}\right) + b.$$

This equation holds according to the description of kernels (4.2.5) with

$$K\left(x^{(i)}, x^{(j)}\right) = \Phi\left(x^{(i)}\right)^T \Phi\left(x^{(j)}\right)$$

and one result of the derivation of the SVM_{DSM} [(23), p.18] using the Lagrange duality:

$$w = \sum_{j=1}^{m} \alpha_j y^{(j)} \Phi\left(x^{(i)}\right).$$

As one can easily see, the **first constraint** of the SVM_{PSM} [(12), p.15] resides inside the brackets of (56). Therefore, the following three locations with respect to the margin (4.2.1) are possible for any vector $x^{(i)}$, $i \in \{1, \ldots, m\}$.

1. The vector $x^{(i)}$ lies **inside the margin** if $y^{(i)} f\left(x^{(i)}\right) < 1$ and $\xi_i > 0$ to satisfy the first SVM_{PSM} constraint. Hence, $\alpha_i = c$ has to hold to satisfy (57).

2. The vector $x^{(i)}$ lies **on the margin** if $y^{(i)} f\left(x^{(i)}\right) = 1$ and $\xi_i = 0$ hold. Hence, (56) and (57) hold for any $\alpha_i \in [0, c]$. In this case, $x^{(i)}$ is called a "support vector".

3. The vector $x^{(i)}$ lies **outside the margin** if $y^{(i)} f\left(x^{(i)}\right) > 1$ and $\xi_i = 0$. Hence, $\alpha_i = 0$ has to hold to satisfy (56).

With these observations, the **input vector's behavior** along the regularization path can be understood. Like mentioned in (4.4.4), the regularization path for a given SVM_{DSM}-pQP [(31), p.23] starts with a sufficiently small parameter c. Additionally, as many as possible vectors $x^{(i)}$ lie inside the margin because $\alpha_i = c$ holds for their associated values α_i. With an increasing parameter c, the value of the objective function increases and the margin's width consequently decreases. Meanwhile, vectors $x^{(i)}$ move from the inside of the margin to the outside while their $\alpha_i = c$ changes to $\alpha_i = 0$ because of the reasons mentioned above. Between the crossing, the vectors $x^{(i)}$ must reside on the margin, meaning that $y^{(i)} f\left(x^{(i)}\right) = 1$ holds. Note that if the orientation of the separating hyperplane (4.2.1) changes, points may move from the inside of the margin to the outside and back again.

While the **criss-cross method** (4.1) varies the parameter c (4.1.4) to compute the entire regularization path, the basis B [(5), p.9] defines sets of vectors $x^{(i)}$ inside, on and outside the margin according to their corresponding values α_i.

The problem is that no definite proposition can be made with respect to the location of any vector $x^{(i)}$ if $\alpha_i = 0$ or $\alpha_i = c$ hold according to the observations above. By concerning the following two statements for SVM_{DSM}-pQPs, this problem is solved:

- If $i \in \{1, \ldots, m\}$ enters the basis B, the variable $\alpha_i = 0$ becomes fixed because $z_i = 0$ holds if $i \in B$ [(6), p.9], where $z^T = \left(\alpha^T, y^T\right)$.
- If $i \in \{m+3, \ldots, 2m+2\}$ leaves the basis B, the variable $\alpha_{i-(m+2)} = c$ becomes fixed because $w_i = 0$ holds if $i \notin B$ [(6), p.9] and consequently $v_{i-m} = -\alpha_{i-(m+2)} + c = 0$ [(3), p.8], where $w^T = \left(u^T, v^T\right)$.

For these reasons, the relation between a fixed basis B and the **location of a vector** $x^{(i)}$ can be described as follows where $i \in \{1, \ldots, m\}$:

- If $i \notin B$ and $m+2+i \notin B$, the variable $\alpha_i = c$ is fixed for the further decrease of the margin's width while varying the parameter c. Hence, $x^{(i)}$ lies **inside the margin**.
- If $i \notin B$ and $m + 2 + i \in B$, no additional limitation beside $0 \leq \alpha_i \leq c$ is given for the further decrease of the margin's width while varying the parameter c. Hence, $x^{(i)}$ lies **on the margin**.
- If $i \in B$ and $m+2+i \in B$, the variable $\alpha_i = 0$ is fixed for the further decrease of the margin's width while varying the parameter c. Hence, $x^{(i)}$ lies **outside the margin**.
- The remaining case, $i \in B$ and $m + 2 + i \notin B$, is not possible because both $\alpha_i = 0$ and $\alpha_i = c$ would have to hold.

It is important to understand the influence of α_i **being variable or fixed** for the location of a vector $x^{(i)}$. According to the observations made in the beginning, it is not certain if the vector $x^{(i)}$ lies inside or on the margin if $\alpha_i = c$ holds, respectively outside or on the margin if $\alpha_i = 0$ holds, without concerning the variability of α_i. Since the vector $x^{(i)}$ may have entered the margin from the inside and left it to the outside afterwards or vice versa, its corresponding α_i must be able to change from $\alpha_i = c$ to $\alpha_i = 0$ linearly because λ_B^* depends linearly on c (4.1.4). This is guaranteed only for vectors $x^{(i)}$ with $i \notin B$ and $m + 2 + i \in B$. For every other possible relation described above, vectors $x^{(i)}$ must definitely either lie outside or inside the margin while varying the parameter c because their corresponding values α_i are fixed.

5 Implementation

This chapter is a collection of **recommendations and tips** we suggest to consider while implementing a solver for pQPs [(2), p.7] using the criss-cross method (4.1). In the following, we assume that the reader is familiar with the **basic concept** of the LU decomposition. Otherwise, a good overview is given in [Her06, chapter 2].

5.1 Data Structures

The solver has to handle the **sparse matrices** M_B and M_N [(5), p.9] while computing $M_B\lambda_B = q$ and $-M_B\Lambda = M_N$ during the criss-cross algorithm (4.1.3). This sparseness is caused by the structure of M [(4), p.8] after transforming an SVM_{DSM} into a pQP (4.3). Therefore, only non-zero elements of M are stored in efficient data structures.

5.1.1 M's Storage

The matrix M is stored **columnwise** because M_B [(5), p.9] needs to be updated with columns of $-M$ in steps 2.1 and 2.2 of the criss-cross algorithm (4.1.3). Additionally, the equation $-M_B\Lambda = M_N$ requires M_N being stored columnwise as described in Hot Spot 2 (5.3.1).

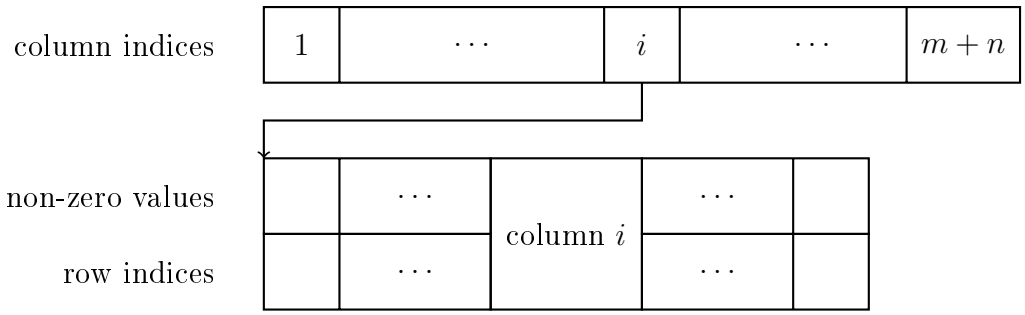

Figure 4: Columnwise data structure of matrix M.

5.1.2 M_B's Rowwise Storage

The matrix M_B [(5), p.9] is stored **rowwise** because its LU decomposition (5.2.1) is more efficient given a rowwise representation. Furthermore, we store the entry with the largest absolute value x_{ij},

$$|x_{ij}| = \max_{k \in \mathbb{K}} |e_{ik}|,$$

where $\mathbb{K}$ is the set of column indices in the active submatrix of M_B at the beginning of each row i. This is done to **avoid searching** for x_{ij} while determining an eligible pivot element (5.2.1). All other entries are stored in arbitrary order afterwards.

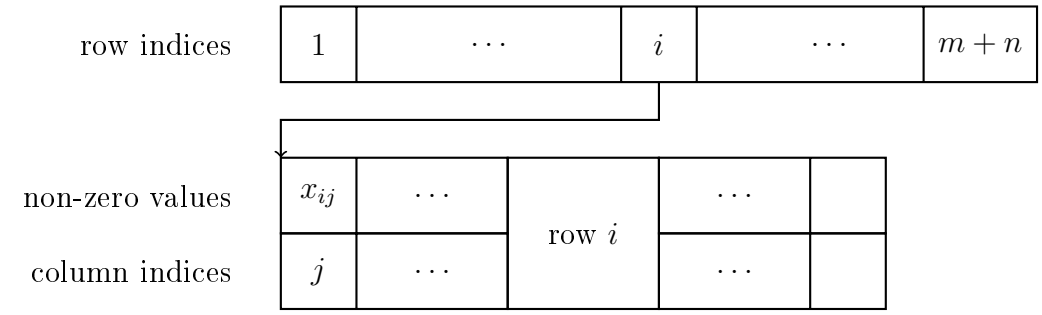

Figure 5: Rowwise data structure of matrix M_B.

5.1.3 M_B's Columnwise Storage

During the **elimination phase** of the LU decomposition (5.2.1), it is timesaving to be able to look up which row in the remaining submatrix has a non-zero entry in the pivot column. Otherwise, the LU decomposition's algorithm would have to search every row in M_B [(5), p.9] for an element in the pivot column to decide whether an elimination is necessary or not.

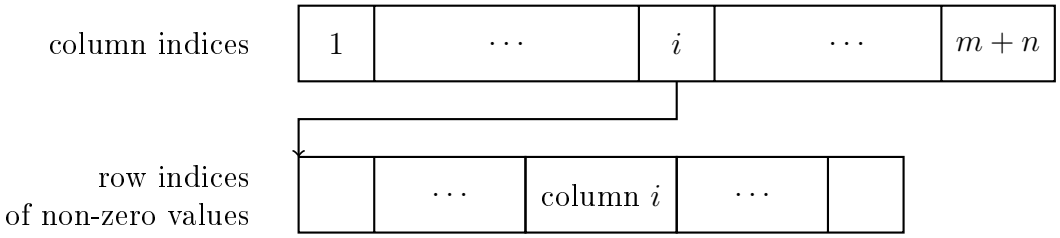

Figure 6: Columnwise data structure of matrix M_B.

5.1.4 Pivot Selection Arrays

Since the pivot element's selection during the LU decomposition (5.2.1) is based on the number of non-zero entries in the rows and columns of the active submatrix, we use the following two data structures to support the organization of these rows and columns while searching for an **eligible pivot element**.

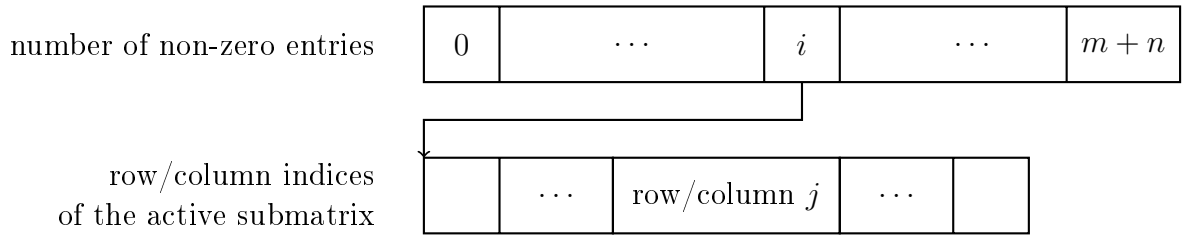

Figure 7: Data structure of row/column indices according to their number of non-zero values in the active submatrix.

We suggest to use **linked lists**[10] **or binary search trees**[11] to administrate the rows respectively columns because the data structures above often have to be updated during the elimination phase of the LU decomposition. These updates are done by searching, deleting and inserting the indices of rows and columns according to their number of non-zero entries in the changing active submatrices.

5.1.5 Active Submatrix Flag

Finally, we recommend to use a **special flag** for each row and column of M_B [(5), p.9] which is set to 0 if the corresponding row or column was already used by a pivot element during the LU decomposition (5.2.1) and to 1 otherwise. Note that all rows and columns carrying a 1-flag define the active submatrix which includes the currently chosen pivot row and column as well.

5.2 Common Techniques

In this subsection, we will demonstrate our approach to compute a basic solution [(6), p.9] and how we handle found diagonal and exchange pivots during the **criss-cross algorithm** (4.1.3) in steps 2.1 and 2.2.

5.2.1 LU Decomposition

During any LU decomposition, the matrix M_B [(5), p.9] is **factorized** into two triangular matrices L and U,

$$Q \cdot M_B \cdot R = L \cdot U,$$

using two permutation matrices Q and R. These row and column permutations of the matrix M_B may be necessary to find eligible pivot elements. We store these transformations "virtually" in **permutation vectors** to access the correct entries in the linear system of

[10] Search in $\mathcal{O}(n)$, insert and delete in $\mathcal{O}(1)$.
[11] Search, insert and delete in $\mathcal{O}(\log n)$.

equations. This is done to protect the rows and columns of M_B from being swapped in the data structures (5.1.2) and (5.1.3).

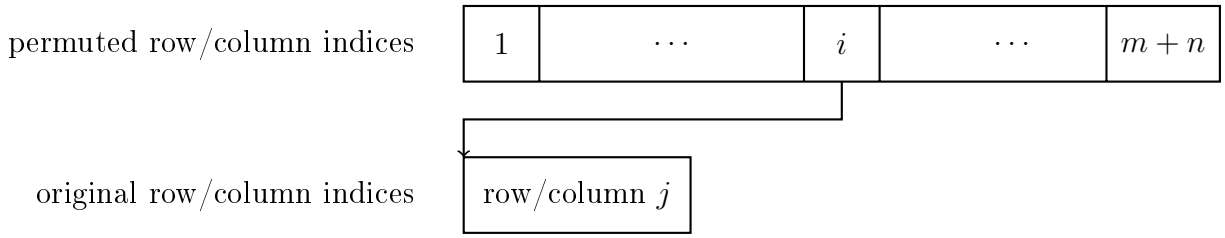

Figure 8: Permutation vectors' data structure.

Starting with the whole matrix M_B, each **round of the LU decomposition** is divided into the pivot selection, additional row and column permutation and the elimination of entries below the found pivot element. Afterwards, both the pivot row and the pivot column remain unchanged for the rest of the LU decomposition which reduces the active submatrix used in the following round.

In the following, we describe **our implementation** of the pivot selection and the standard elimination phase:

Pivot Selection: This part of the LU decomposition is **very critical** because it has a great influence on the overall accuracy, the density of L and U and the running time.

Since we rather had problems with the accuracy than with the running time, we decided to implement an approach based on [SS90] but using a modified **total pivoting** strategy. Given a value $0 \leq u \leq 1$, the algorithm works as follows:

1 | **IF** a column with only one non-zero entry is available in the active submatrix **THEN** choose it as pivot element;

2 | **ELSE IF** a row with only one non-zero entry is available in the active submatrix **THEN** choose it as pivot element;

3 | **ELSE**

3.1 | **IF** $u \neq 0$ **THEN** determine the entry x_{ij} with the largest absolute value among all entries in the active submatrix,

$$|x_{ij}| = \max_{l \in \mathbb{L}, k \in \mathbb{K}} |e_{lk}|,$$

where $\mathbb{L}$ is the set of all row indices and $\mathbb{K}$ is the set of all column indices in the active submatrix of M_B;

3.2 | **IF** $u = 1$ **THEN** choose x_{ij} as pivot element;

3.3 | **ELSE FOR** $h = 2$ **TO** $m + n$ **DO**

3.3.1 | **IF** a column k with h non-zero entries is available in the active submatrix **AND**

$$|p_{lk}| = \max_{l \in \mathbb{L}} |e_{lk}| \geq u \cdot |x_{ij}|$$

holds **THEN** choose p_{lk} as pivot element;

3.3.2 | **ELSE IF** a row l with h non-zero entries is available in the active submatrix **AND**

$$|p_{lk}| = \max_{k \in \mathbb{K}} |e_{lk}| \geq u \cdot |x_{ij}|$$

holds **THEN** choose p_{lk} as pivot element;

As one can easily see, steps 1, 2 and 3.3 are taken from [SS90, p. 329] to preserve the **sparsity** of the LU decomposition, where the choice of the pivot element in steps 1 and 2 has little if any effect on the further accuracy.

In contrast to [SS90], the choice of the value u in combination with the **stability criterion** used in steps 3.3.1 and 3.3.2 adjusts the focus from partial pivoting for $u = 0$ to total pivoting for $u = 1$. Like suggested in [SS90, subsection 2.3], we use the default value $u = 10^{-2}$, where numerical stability is guaranteed only for $u = 1$ as mentioned in [SS90, subsection 2.2]. Hence, our algorithm focuses its attention on increasing the accuracy rather than preserving the sparsity of the LU decomposition. This approach reduces the number of iterations using the iterative refinement (5.2.3) to compute results which are accurate enough.

After a pivot element is chosen, the **pivot row and column** are added to the permutation vectors and deleted from the data structures recording the number of non-zero elements in the active submatrix (5.1.4).

Note that **not much effort** is necessary to determine x_{ij} in step 3.1 because the element of the largest absolute value of every row is stored at the row's beginning (5.1.2).

Elimination: By using the data structure storing the row indices for each column (5.1.3), all rows affected by the pivot column are determined and the standard elimination is performed, see for example [Her06, chapter 2]. In this process, both triangular matrices L and U are stored in M_B's data structure to reduce the **memory requirements**.

During this phase, it may be necessary to decide whether a computed value in a non-pivot row affected by the elimination is equal to zero or not. As suggested in [SS90, subsection 2.3], we control the stability of our implementation by using the **drop tolerance** $c = 10^{-9}$ where any computed value x is set to zero if $|x| < c$.

Furthermore, we monitor the stability of the LU decomposition and cancel it if the largest absolute value exceeds 10^{15}. Remember that choosing a larger value u in the pivot selection can increase the stability.

In the **end**, it has to be assured that the entry with the largest absolute value is stored as first element of each affected row as described in (5.1.2). Additionally, M_B's columnwise data structure (5.1.3) as well as the arrays recording the number of non-zero entries of the active submatrix (5.1.4) have to be updated for all changed rows and columns. Finally, the special flag (5.1.5) is set to -1 for the pivot row and column.

5.2.2 Solving Systems of Linear Equations

We use the **LU decomposition** (5.2.1) of the matrix M_B [(5), p.9] to compute $M_B\lambda_B = q$ and $-M_B\Lambda = M_N$ during the criss-cross algorithm (4.1.3) and to determine the largest valid $\hat{\mu}$ while varying the regularization parameter (4.1.4).

In the following, the computation of $Q \cdot M_B \cdot RR^{-1} \cdot \lambda_B = Q \cdot q$ is described, where $Q \cdot M_B \cdot R = L \cdot U$ holds and Q,R are **transformation matrices** for row and column permutations.

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & & \\ l_{31} & l_{32} & 1 & & \vdots \\ \vdots & & & \ddots & \\ l_{m+n,1} & & \cdots & & 1 \end{pmatrix} \cdot \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_{m+n} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{m+n} \end{pmatrix}$$

$$\begin{aligned} z_1 &= b_1 \\ z_2 &= b_2 - l_{21} \cdot z_1 \\ z_3 &= b_3 - l_{31} \cdot z_1 - l_{32} \cdot z_2 \\ &\cdots \end{aligned}$$

Figure 9: Step 1: solving $L \cdot z = Q \cdot q = b$.

At first, vector z is computed **top-down** by going through all rows of the matrix L step by step using the precomputed entries in z to determine the products $l_{ij} \cdot z_j$, $i, j \in \{1, \ldots, k\}$ with $k = m + n$.

$$\begin{pmatrix} u_{11} & & \cdots & & u_{1,k} \\ & \ddots & & & \vdots \\ \vdots & & u_{k-2,k-2} & u_{k-2,k-1} & u_{k-2,k} \\ & & & u_{k-1,k-1} & u_{k-1,k} \\ 0 & & \cdots & & u_{k,k} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_{k-2} \\ x_{k-1} \\ x_k \end{pmatrix} = \begin{pmatrix} z_1 \\ \vdots \\ z_{k-2} \\ z_{k-1} \\ z_k \end{pmatrix}$$

$$\begin{aligned} & \cdots \\ x_{k-2} &= (z_{k-2} - u_{k-2,k-1} \cdot x_{k-1} - u_{k-2,k} \cdot x_k)/u_{k-2,k-2} \\ x_{k-1} &= (z_{k-1} - u_{k-1,k} \cdot x_k)/u_{k-1,k-1} \\ x_k &= z_k/u_{k,k} \end{aligned}$$

Figure 10: Step 2: solving $U \cdot R^{-1} \cdot \lambda_B = U \cdot x = z$.

Second, the computation of the vector x is designed similarly to the first step but in a **bottom-up** fashion.

As described in (5.2.1), both row and column permutation are "virtual", meaning that **permutation vectors** record Q and R to access the correct entries in M_B while its data structures in (5.1.2) and (5.1.3) remains unchanged.

Remember that the criss-cross algorithm in step 2 has to choose the smallest index $r \in \{1, \ldots, m+n\}$ so that $(\lambda_B^*)_r < 0$ holds. Thus, the **original sequence** of the vector λ_B^* has to be determined by switching the indices of vector x:

$$\lambda_B^* = Rx.$$

5.2.3 Iterative Refinement

After solving a system of linear equations (5.2.2), it may be necessary to improve the solution's accuracy (5.3) because of the **restricted machine accuracy**. Therefore, we use the iterative refinement which is described in the following according to [Her06, subsection 2.3]:

Definition: Let $x \in \mathbb{R}^n$ be the exact solution of the system of equations $Ax = b$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$. Let further $\tilde{x} \in \mathbb{R}^n$ be an approximation of x. The **residual** $r(\tilde{x}) \in \mathbb{R}^n$ belonging to $\tilde{x}$ is defined as follows:

$$r(\tilde{x}) := b - A\tilde{x} = A(x - \tilde{x}).$$

In addition, let $x^{(0)} := \tilde{x}$ be the computed approximation of x. Then, $\Delta x^{(0)} \in \mathbb{R}^n$ is the **patch-vector** so that

$$x = x^{(0)} + \Delta x^{(0)}$$

holds. Plugging this into $Ax = b$ results in

$$A\Delta x^{(0)} = b - Ax^{(0)} = r\left(x^{(0)}\right). \tag{58}$$

Since the result $\Delta x^{(0)}$ of the equation has numerical rounding errors too, only an approximation $\tilde{\Delta} x^{(0)}$ can be achieved. Nevertheless,

$$x^{(1)} := x^{(0)} + \tilde{\Delta} x^{(0)}$$

will be a better approximation of x than $x^{(0)}$. The iterative refinement **repeats this procedure** until the solution is approximated accurately enough. Because the matrix A remains unchanged during the refinement, its already computed LU decomposition can be used to determine all $\Delta x^{(i)}$, $i \in \mathbb{N}_0$.

According to [SK06, p. 56], it can be expected that **each new round** of the iterative refinement will determine as many additional valid digits [(59), p.51] as have been valid in the round before.

5.2.4 Diagonal and Exchange Pivot

Here, the focus lies on **efficient methods** of handling found diagonal and exchange pivots as mentioned in the criss-cross algorithm (4.1.3) in steps 2.1 and 2.2. In the following, two different approaches are presented:

- A simple method is to **recreate the matrix** $M_{B'}$ [(5), p.9] according to the updated basis B'. This is necessary because the data structures of M_B, (5.1.2) and (5.1.3), are overwritten with its LU decomposition (5.2.1). Thus, both data structures of the matrix M_B are cleared and the special variable mentioned in (5.1.5) is reset to -1 for all rows and columns. Afterwards, the matrix $M_{B'}$ is constructed as described in [(5), p.9] and stored both row- and columnwise again.
- More complex methods try to **update the LU decomposition** of the matrix M_B with the new columns according to B' as fast as possible. Furthermore, their focus lies on maintaining the sparsity and the numerical stability of the resulting LU decomposition of the matrix $M_{B'}$. There exist various techniques of LU updates like [BG69], [FT72], [Rei82] and [SS93].

Since $M_{B'}$ is restored with the original entries of the matrices $-M$ and I and because the LU decomposition is computed every round of the criss-cross algorithm from scratch, the simple method has the advantage of being the best with regard to the **numerical**

stability. Thus, no error propagation occurs and the sparsity of the LU decomposition of $M_{B'}$ only depends on the pivot selection. On the other hand, this computation is adverse concerning its running time because it is done without using any knowledge of the precedent LU decomposition of M_B in contrast to the more complex methods.

Since we had more problems with the accuracy than with the running time, we decided to implement the simple method. An **alternative** is to use a method that updates the LU decomposition of the matrix M_B and to monitor both sparseness and numerical stability. If one of them reaches a critical level, the simple method is applied once to correct the problem.

5.3 Accuracy Hot Spots

Since the criss-cross method (4.1) is non-approximate, it has to deal with the problem of depending on accurate computations despite the **restricted machine accuracy**. Thus, the determination of useful results is an important subarea of the implementation and has to be handled with care. The following definitions are necessary to describe our approach in this field:

Definition: Let $x^{(i)} \in \mathbb{R}^n$ be the approximation of $x \in \mathbb{R}^n$ during the i-th round of an iterative refinement (5.2.3). Given the patch-vector $\tilde{\Delta}x^{(i)}$ [(58), p.50], the number of **valid digits** of an entry $x_r^{(i)}$ is given by

$$\mathrm{vd}^{(i)}\left(x_r^{(i)}, \tilde{\Delta}x_r^{(i)}\right) := \begin{cases} \left\lfloor \log_{10}\left|x_r^{(i)}\right| - \log_{10}\left|\tilde{\Delta}x_r^{(i)}\right| \right\rfloor & , \; x_r^{(i)} \neq 0 \wedge \tilde{\Delta}x_r^{(i)} \neq 0 \\ \left\lfloor -\log_{10}\left|\tilde{\Delta}x_r^{(i)}\right| \right\rfloor & , \; x_r^{(i)} = 0 \wedge \tilde{\Delta}x_r^{(i)} \neq 0 \\ \infty & , \; \text{otherwise} \; . \end{cases} \tag{59}$$

Notation: Given a value $x \in \mathbb{R}$ in decimal notation. With $x_{[k]}$, $k \in \mathbb{N}\backslash\{0\}$, we describe the k-th digit of x. For example, $x = [0,04711]_{10}$ has the following digits:

$$x_{[1]} = 0, \; x_{[2]} = 0, \; x_{[3]} = 4, \; x_{[4]} = 7, \; x_{[5]} = 1, \; x_{[6]} = 1, \; x_{[7]} = 0, \; \ldots$$

The **basic concept** is to define a minimal number $a \in \mathbb{N}\backslash\{0\}$ of valid digits to be determined for any accuracy hot spot to obtain useful results. The choice of a is difficult and depends on empirical tests. If a is too small, errors will occur during the computation because decisions are made on inaccurate results. Otherwise, too much time is spent on determining results with more valid digits than necessary. Note that we use $a = 8$ in our IEEE 754 double precision implementation which is a suitable empirical value for our problems.

5.3.1 Criss-Cross Algorithm

The first **two hot spots** are located in the criss-cross algorithm (4.1.3) where inequations have to be checked against zero. The difficulty lies in deciding if a value $x \in \mathbb{R}$ is equal to zero with respect to the restricted machine accuracy which can be handled for the different hot spots as follows:

Hot Spot 1: $\lambda_B^* < 0$, criss-cross algorithm step 1.

Since vector λ_B^*'s accuracy affects the third hot spot (5.3.2) as well, λ_B^* has to be computed as accurately as possible. Given a minimal number $a \in \mathbb{N}\backslash\{0\}$ of valid digits to be determined, we recommend to run the **iterative refinement** (5.2.3) of λ_B^* as long as one of the following two cases occurs:

1. Vector λ_B^* can be approximated **accurately enough**, meaning that

$$\mathrm{vd}^{(i)}\left(\left(\lambda_B^*\right)_r^{(i)}, \tilde{\Delta}\left(\lambda_B^*\right)_r^{(i)}\right) \geq a$$

   holds for all $r \in \{1, \ldots, m+n\}$ after the i-th round. In this case, the value of a is not changed.

2. At least one entry $\left(\lambda_B^*\right)_r$ cannot be approximated accurately enough, meaning that

$$\mathrm{vd}^{(i)}\left(\left(\lambda_B^*\right)_r^{(i)}, \tilde{\Delta}\left(\lambda_B^*\right)_r^{(i)}\right) < a$$

   holds for all $i \in \mathbb{N}_0$. This characteristic can occur if $\tilde{\Delta}\left(\lambda_B^*\right)_r^{(i)}$ starts to alternate because of the restricted machine accuracy. In this case, further rounds of the iterative refinement do not lead to a better approximation of $\left(\lambda_B^*\right)_r$. Thus, a has to be **decreased** to satisfy the first case. In doing so, some of the already computed rounds of the iterative refinement become worthless. This affects at least the rounds necessary to identify the alternation of $\tilde{\Delta}\left(\lambda_B^*\right)_r^{(i)}$.

   Note that a is reset to its defined value if and only if $M_B\lambda_B = q$ is solved for the next time. Thus, a monitors the **global accuracy** during one round of the criss-cross algorithm and helps to avoid running into the second case too often which would decrease the running time.

After one of the two cases occurs in the i-th round of the iterative refinement of λ_B^*, we consider any entry $\left(\lambda_B^*\right)_r^{(i)}$, $r \in \{1, \ldots, m+n\}$ as **smaller than zero** if both

$$\mathrm{sgn}\left(\left(\lambda_B^*\right)_r^{(i)}\right) = -1 \quad \text{and} \quad \left(\left(\lambda_B^*\right)_r^{(i)}\right)_{[k]} > 0$$

hold for any $k \in \{1, \ldots, a\}$.

Hot Spot 2: $\Lambda_{rj} \geq 0$, criss-cross algorithm steps 2.1 and 2.2.

Let further A_j be the j-th columns and A^r be the r-th row of any matrix A. In the following, **two different approaches** are described to determine useful results Λ_{rj}.

- The simplest approach is to compute the elements of Λ^r and their iterative refinements **columnwise**,

$$-M_B \Lambda_j = (M_N)_j \,,$$

where $j \in \{1, \ldots, m+n\}$, starting with the r-th column according to the criss-cross algorithm in step 2.1. Since no column of Λ is of further use, the focus of the iterative refinement lies only on the elements of Λ^r. As described in Hot Spot 1, two cases are possible:

1. The entry Λ_{rj} can be approximated **accurately enough**, meaning that

$$\mathrm{vd}^{(i)}\left(\Lambda_{rj}^{(i)}, \tilde{\Delta}\Lambda_{rj}^{(i)}\right) \geq a$$

holds after the i-th round. In this case, the value of a is not changed.

2. The entry Λ_{rj} cannot be approximated accurately enough, meaning that

$$\mathrm{vd}^{(i)}\left(\Lambda_{rj}^{(i)}, \tilde{\Delta}\Lambda_{rj}^{(i)}\right) < a$$

holds for all $i \in \mathbb{N}_0$. Hence, a has to be **decreased**.

Nevertheless, this columnwise procedure becomes **inefficient** every time $\Lambda_{rr}^{(i)}$ is considered to be equal to zero which is if

$$\left(\Lambda_{rr}^{(i)}\right)_{[k]} = 0$$

holds for all $k \in \{1, \ldots, a\}$. This inefficiency is caused by further **columnwise computations** for a single element while determining the smallest index s so that $\Lambda_{rs} > 0$ holds.

- A more **sophisticated approach** is to transform $\Lambda^r = \left(-M_B^{-1}\right)^r M_N$ into

$$\left(\Lambda^T\right)_r = M_N^T \left(-M_B^T\right)_r^{-1} . \tag{60}$$

With $M_B^{-1} M_B = I$, it follows that

$$M_B^T \left(M_B^T\right)_r^{-1} = I_r \tag{61}$$

holds, where I is the $k \times k$ identity matrix. Hence, the necessary r-th column $\left(-M_B^T\right)_r^{-1}$ for (60) can be computed using the **LU decomposition** (5.2.1) of the matrix M_B by solving

$$R^T \cdot M_B^T \cdot Q^T \left(Q^T\right)^{-1} \cdot \left(M_B^T\right)_r^{-1} = R^T \cdot I_r,$$

where $R^T \cdot M_B^T \cdot Q^T = U^T L^T$ holds, similarly to the description in (5.2.2). Note that U^T is a lower and L^T is an upper triangular matrix and that M_B is **invertible** because of B being a basis [(5), p.9]. Additionally, M_N is addressed columnwise as stored (5.1.1) in (60).

The following algorithm **increases the accuracy** of the approximation $\left(\Lambda^T\right)_r^{(0)}$ of $\left(\Lambda^T\right)_r$ in (60). It is implemented according to the criss-cross algorithm's requirements in steps 2.1 and 2.2 by using the iterative refinement and starting with $i = 0$ and $j = r$:

1 | Compute one round of the iterative refinement of $\left(M_B^T\right)_r^{-1}$ using (61) and resulting in the patch-vector [(58), p.50]

$$\tilde{\Delta}\left(\left(M_B^T\right)_r^{-1}\right)^{(i)};$$

2 | Compute the patch-vector for $\left(\Lambda^T\right)_r^{(i)}$ using (60) with

$$\tilde{\Delta}\left(\Lambda^T\right)_r^{(i)} = M_N^T \cdot \tilde{\Delta}\left(\left(-M_B^T\right)_r^{-1}\right)^{(i)};$$

3 | **IF** $\left(\Lambda^T\right)_{jr}$ is approximated accurately enough, meaning that

$$\mathrm{vd}^{(i)}\left(\left(\Lambda^T\right)_{jr}^{(i)}, \tilde{\Delta}\left(\Lambda^T\right)_{jr}^{(i)}\right) \geq a,$$

THEN handle the exchange or diagonal pivot or continue with step 3 for another index j according to the criss-cross algorithm's requirements. The value $\left(\Lambda^T\right)_{jr}^{(i)}$ is considered as larger than zero if both

$$\mathrm{sgn}\left(\left(\Lambda^T\right)_{jr}^{(i)}\right) = 1 \quad \text{and} \quad \left(\left(\Lambda^T\right)_{jr}^{(i)}\right)_{[k]} > 0$$

hold for any $k \in \{1, \ldots, a\}$;

4 | **ELSE** set $i = i + 1$, determine $\left(\Lambda^T\right)_r^{(i)}$ and $\left(\left(M_B^T\right)_r^{-1}\right)^{(i)}$ by adding the patch-vectors computed in steps 1 and 2, repeat the algorithm or decrease a as described in Hot Spot 1;

5.3.2 Varying the Regularization Parameter

As described in (4.1.4), the **largest valid** $\hat{\mu}$, where $\mu' := \mu + \hat{\mu} \geq \mu$ so that $\lambda_B^*(\mu') \geq 0$ holds, is given by [(9), p.11]:

Hot Spot 3: $\hat{\mu} = \min \left\{ \frac{\left(-\lambda_B^*(\mu)\right)_r}{\delta_r} \mid \delta_r < 0 \right\}$,

where δ_r is defined by $M_B \cdot \delta = d$. In our implementation, we place the focus on computing $\mu' + \varepsilon = \mu + \hat{\mu} + \varepsilon$ because the criss-cross method (4.1) has to be applied on the pLCP [(4), p.8] using $q(\mu' + \varepsilon)$ again to compute the entire **regularization path**. Therefore, the modified equation,

$$-10^{-a_{def}+1} = (\hat{\mu} + \varepsilon)\,\delta + \lambda_B^*(\mu),$$

where a_{def} is the originally defined value a in (5.3), has to be solved for every negative entry in δ to determine the largest valid

$$\hat{\mu} + \varepsilon = \min \mathcal{D},\ \mathcal{D} := \left\{ \frac{-10^{-a_{def}+1} - \left(\lambda_B^*(\mu)\right)_r}{\delta_r} \mid \delta_r < 0 \right\}. \tag{62}$$

Remember that the value a defining the minimal number of **valid digits** is reset to its defined value a_{def} if and only if the criss-cross method is applied again and solves $M_B \lambda_B(\mu' + \varepsilon) = q(\mu' + \varepsilon)$ as described in Hot Spot 1 (5.3.1). In this case, any entry $(\lambda_B^*(\mu' + \varepsilon))_r^{(i)}$, $r \in \{1, \ldots, m+n\}$ is considered as smaller than zero if both

$$\operatorname{sgn}\left((\lambda_B^*(\mu' + \varepsilon))_r^{(i)}\right) = -1 \quad \text{and} \quad \left((\lambda_B^*(\mu' + \varepsilon))_r^{(i)}\right)_{[k]} > 0$$

hold for any $k \in \{1, \ldots, a_{def}\}$ given that a_{def} is not decreased as described in Hot Spot 1. Hence, the choice of $-10^{-a_{def}+1}$ (62) instead of zero in [(9), p.11] becomes clear: In step 2, the reapplied criss-cross algorithm (4.1.3) **correctly recognizes** the entry $(\lambda_B^*(\mu' + \varepsilon))_r = -10^{-a_{def}+1}$ with

$$\operatorname{sgn}\left((\lambda_B^*)_r^{(i)}\right) = -1 \quad \text{and} \quad \left((\lambda_B^*)_r^{(i)}\right)_{[a_{def}]} = 1 \tag{63}$$

as smaller than zero.

Nevertheless, it is possible that a_{def} has to be decreased because of the restricted machine accuracy while computing $M_B \lambda_B(\mu' + \varepsilon) = q(\mu' + \varepsilon)$ as described in Hot Spot 1. This leads to the problem that the criss-cross algorithm in step 2 cannot find the negative entry (63) although $\mu' + \varepsilon \neq \infty$. Therefore, we **pass the index** r of

$$\frac{-10^{-a_{def}+1} - \left(\lambda_B^*(\mu)\right)_r}{\delta_r} \in \mathcal{D}$$

defining the largest valid $\hat{\mu} + \varepsilon$ in (62) to the reapplied criss-cross algorithm in step 2 as the smallest index r to avoid searching for it. The problem is that $\hat{\mu} + \varepsilon$ can be defined by **two or more** elements of $\mathcal{D}$,

$$\left| \left\{ r \mid \frac{-10^{-a_{def}+1} - \left(\lambda_B^*(\mu)\right)_r}{\delta_r} = \hat{\mu} + \varepsilon \right\} \right| > 1.$$

The decision whether $x = y = \hat{\mu} + \varepsilon$ holds, where $x, y \in \mathcal{D}$, depends on the accuracy of **both vectors** $\lambda_B^*(\mu)$ and δ.

- $\lambda_B^*(\mu)$ is already approximated accurately enough using the approach in Hot Spot 1.
- δ has to be approximated in analogy to Hot Spot 1.

Additionally, the following **residual** [(58), p.50] of x,

$$r\left(x^{(i)}\right) = -10^{-a_{def}+1} - \left(\lambda_B^*(\mu)\right)_r - \delta_r \cdot x^{(i)} = \delta_r \cdot \tilde{\Delta} x^{(i)},$$

where y is handled in the same way, is used to compute the iterative refinement as long as x is approximated **accurately enough**, meaning that

$$\mathrm{vd}^{(i)}\left(x^{(i)}, \tilde{\Delta} x^{(i)}\right) \geq a$$

holds after the i-th round. Here, a is the value possibly decreased by **former adjustments** during the criss-cross algorithm (5.3.1). If x cannot be approximated accurately enough, meaning that

$$\mathrm{vd}^{(i)}\left(x^{(i)}, \tilde{\Delta} x^{(i)}\right) < a$$

holds for all $i \in \mathbb{N}_0$, a has to be decreased as described in Hot Spot 1.

Given two values $x, y \in \mathcal{D}$ approximated accurately enough, both values are equal if and only if $x_{[k]} = y_{[k]}$ holds for all $k \in \{1, \ldots, a\}$ as well as $\lfloor \log_{10} x \rfloor = \lfloor \log_{10} y \rfloor$, meaning that both values x and y have the **decimal mark** at the same position. Note that if a is decreased between the computations of x and y, the smaller a has to be taken because the accuracy of their comparison is limited by the more inaccurate value.

Finally, $\mu' = \mu + \hat{\mu}$ has to be computed to specify the **whole interval** $[\mu, \mu']$ for which B [(5), p.9] is valid and $\lambda_B^*(\kappa)$ is a solution to the pLCP with the right-hand side $q(\kappa)$ for every value $\kappa \in [\mu, \mu']$. This is achieved by using the equation

$$\hat{\mu} = \frac{\left(-\lambda_B^*(\mu)\right)_r}{\delta_r}$$

with the determined index r of

$$\frac{-10^{-a_{def}+1} - \left(\lambda_B^*(\mu)\right)_r}{\delta_r} \in \mathcal{D}$$

defining the largest valid $\hat{\mu} + \varepsilon$.

5.3.3 Basis Reconstruction - General Case

Remember the following **dual pQP** [(37), p.26] which has to be solved to reconstruct the basis B in the general case (4.4.1):

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^m} \quad & -\tfrac{1}{2} x^T Q x + y^T b(\mu) \\ \text{s.t.} \quad & A^T y \leq Q^T x + c(\mu) \\ & y \geq 0 \ . \end{aligned}$$

As suggested, we use the proprietary software **IBM ILOG CPLEX** to solve both the SVM_{DSM}-pQP [(31), p.23] and the dual pQP for a fixed parameter μ.

Hot Spot 4: x and y.

Since the solution x of the SVM_{DSM}-pQP has to be computed before the dual pQP can be solved, the accuracy of x affects how accurately y can be determined. Furthermore, the **basis reconstruction** in the general case is also based on the vectors u and v, where

$$u = c(\mu) - A^T y + Qx \quad \text{and} \quad v = Ax - b(\mu) \, .$$

Thus, we **highly recommend** to determine both vectors x and y as accurately as possible depending on the chosen solver. Additionally, attention has to be paid if data is written to and read from the chosen solver because conversations can have a negative influence on the accuracy of x and y.

5.4 Data Formats

As mentioned in (4.4.2), we use the free software LIBSVM [CCCJ01] to estimate the value c at the beginning of the regularization path. Therefore, we suggest to record all pattern vectors $x^{(i)}$ and their corresponding labels $y^{(i)}$ in the **LIBSVM data format** which stores the data in a sparse representation.

$y^{(1)} = -1,\ x_{17}^{(1)} = -1,\ x_{18}^{(1)} = 1$	-1 17:-1 18:1
$\cdots$	-1 1:1 3:-1 7:-1 8:1 9:-1 10:1 15:1 16:-1
	+1 2:-1 3:1 7:1 8:-1 10:-1 11:1 17:1 20:-1
	+1 6:1 7:-1 10:1 11:-1 18:-1 20:1
	...

Figure 11: Example of the LIBSVM data format recording a training set $\mathcal{T}$ (4.2.1).

Moreover, we use the proprietary software CPLEX to solve the SVM_{DSM}-pQP [(31), p.23] and the corresponding dual pQP [(37), p.26], both for a fixed parameter μ as described in

(4.4.1). As input data for CPLEX, we recommend to convert both problems into the **LP file format** which can easily be read and created.

	Minimize
quadratic part $\frac{1}{2}\alpha^T Q\alpha$	obj: [+ 1.00000000000000000000 a0 * a0
	+ 0.60653065971263342000 a0 * a1
	...
	+ 0.54881163609402639000 a899 * a898
	+ 1.00000000000000000000 a899 * a899] / 2
linear part $\hat{c}^T\alpha$	- a0 ... - a899
	Subject To
constraint $A\alpha \geq b(c)$	c0: - 1 a0 - 1 a1 + 1 a2 + 1 a3 ... - 1 a899 => 0
	c1: + 1 a0 + 1 a1 - 1 a2 - 1 a3 ... + 1 a899 => 0
	c2: - 1 a0 => - 1
	...
	c901: - 1 a899 => - 1
	c902: a0 => 0
constraint $\alpha \geq 0$	...
	c1801: a899 => 0
	End

Figure 12: Example of the CPLEX LP file format recording an SVM_{DSM}-pQP.

After using CPLEX as mentioned above, we prefer the **SOL file format** as output data because the solution's extraction can be implemented with little effort.

```
solution values
<?xml version = "1.0" standalone="yes"?>
<CPLEXSolution version="1.2">
  <header ... />
  <quality ... />
  <linearConstraints> ... </linearConstraints>
  <variables>
    <variable name="a0" ... value="0.999999999999996" .../>
    ...
    <variable name="a899" ... value="7.99588939605571e-14" .../>
</variables>
</CPLEXSolution>
```

Figure 13: Example of the SOL file format recording the solution of an SVM_{DSM}-pQP.

Beside the presented representations, **other formats** can be used together with CPLEX like described in [IBM09].

5.5 Further Implementation Aspects

5.5.1 Parallel Computing

While working on our implementation, we tried to parallelize the code in various ways. The problem is that the criss-cross method (4.1) uses an iterative type algorithm where every round depends on former results. Thus, parallelism can only reside **within a single round** of the criss-cross algorithm (4.1.3), for example in determining λ_B^* and Λ_r or in updating M_B [(5), p.9].

As shown in the experimental results (5.6), the criss-cross algorithm needs a large number of rounds to compute the entire regularization path by varying the parameter μ (4.1.4). Since one round is already very fast in our problem setups, the running times of all tested parallel variants exceed the iterative implementation's running time because of the **parallel overhead**.

Therefore, we recommend to place the focus on finding a **better start basis** compared to B_{def} (4.1.3) rather than optimizing the code using parallel computing. As mentioned in (4.4.3), this approach depends on the pQP's structure and requires a deep understanding of the corresponding geometry (4.6).

Nevertheless, parallel computing can increase the performance of the implementation outside the criss-cross algorithm: We use a **prediction set** $\mathcal{P}$ (4.4.2) to compute the prediction accuracy [(40), p.28] along the regularization path. This prediction can easily be parallelized because each date $\left(x^{(i)}, y^{(i)}\right) \in \mathcal{P}$ can be handled independently.

5.5.2 Sherman-Morrison Formula

As we tested **different implementations** of pQP-solvers [(2), p.7] using the criss-cross method (4.1), we also tried using the well-known Sherman-Morrison formula, see for example [Her06, p. 71]. The following definition of this formula is correspondingly[12] taken from [Hig08, p. 329].

Definition: If $A \in \mathbb{R}^{n \times n}$ is nonsingular and $v^T A^{-1} u \neq -1$ then $A + uv^T$ is nonsingular and

$$\left(A + uv^T\right)^{-1} = A^{-1} - \frac{\left(A^{-1}u\right)\left(v^T A^{-1}\right)}{1 + v^T A^{-1} u}. \tag{64}$$

[12] No changes with regard to content; only modifications in notation and highlighting.

In contrast to the description of the implementation so far, the following approach can avoid the need for computing the **LU decomposition** (5.2.1) of M_B [(5), p.9] during the criss-cross method: If the criss-cross algorithm (4.1.3) starts with the default start basis B_{def} (4.1.3), $M_B = M_B^{-1} = I$ holds. Otherwise, the inverse matrix M_B^{-1} has to be computed. Following the criss-cross algorithm, both necessary calculations are given by $\lambda_B = M_B^{-1}q$ and $\Lambda = -M_B^{-1}M_N$. For both the diagonal and the exchange pivot in steps 2.1 and 2.2 of the criss-cross algorithm, (64) can be used to compute $M_{B'}^{-1}$, where M_B is being updated to $M_{B'}$ with uv^T describing $B \oplus \{r\}$ and $B \oplus \{s\}$ if necessary.

The problem of this approach is that the criss-cross algorithm is an iterative method, resulting in a **high error propagation** if only (64) is used to compute $M_{B'}^{-1}$. To eliminate this disadvantage, the well-known condition number

$$\operatorname{cond}(A) = \|A\| \left\|A^{-1}\right\|,$$

see for example [Her06, p. 86], can be used to monitor the accuracy: As an increasing **condition number** is a sign of the problem becoming ill-conditioned, it can initialize a recalculation of the matrix $M_{B'}^{-1}$ from scratch by solving $M_{B'}M_{B'}^{-1} = I$ if a threshold is reached. The determination of this threshold depends on the concrete accuracy requirements of the problem setup and has to be done empirically.

5.5.3 The General Case

As the width of the margin decreases and pattern vectors $x^{(i)}$ (4.2.1) move from the inside of the margin to the outside while increasing parameter c (4.6), the geometry of our problem setups leads to a **continuous behavior**. Hence, we have no need of implementing the general case as described in (4.1.5) in our solver.

Nevertheless, the **implementation** of the general case can be handled in analogy to varying the regularization parameter (5.3.2). If an unsolvable situation occurs, meaning that $(\lambda_B^*(\mu))_r < 0$, $\Lambda_{rr} = 0$ and $\Lambda_{rj} \leq 0$ hold for all $j \in \{1, \ldots, m+n\}$, μ has to be adjusted to $\mu' = \mu + \hat{\mu} \geq \mu$ so that $(\lambda_B^*(\mu'))_r \geq 0$ holds. To determine the value $\hat{\mu}$, the equation

$$\hat{\mu} = \frac{(-\lambda_B^*(\mu))_r}{\delta_r}$$

has to be solved, where the index r is already given through $(\lambda_B^*(\mu))_r < 0$. Note that it is impossible to find an eligible value $\hat{\mu} \geq 0$ and that the pLCP [(4), p.8] is **definitely unsolvable** if $\delta_r \leq 0$ holds.

5.5.4 Handling CPLEX and LIBSVM

In this subsection, we demonstrate how we use both the proprietary software CPLEX in (4.4.1) for basis reconstruction and the free software LIBSVM in (4.4.2) to determine the regularization path's start.

CPLEX: In the following, we describe the use of CPLEX to solve a **QP** given in the LP file format (5.4). After running the CPLEX executable, we determine the QP's solution in the following way:

```
CPLEX> set barrier convergetol 1e-012
New value for complementarity tolerance: 1e-012

CPLEX> read QP.lp
Problem 'QP.lp' read.
Read time =    2.29 sec.

CPLEX> optimize
Number of nonzeros in lower triangle of Q = 1279200
Using Approximate Minimum Degree ordering
Total time for automatic ordering = 0.23 sec.
Summary statistics for factor of Q:
  Rows in Factor            = 1600
  Integer space required    = 1600
  Total non-zeros in factor = 1280800
  Total FP ops to factor    = 1366613600
Tried aggregator 1 time.
QP Presolve eliminated 3261 rows and 83 columns.
Reduced QP has 1541 rows, 3117 columns, and 1212725 nonzeros.
Reduced QP objective Q matrix has 1540 nonzeros.
Presolve time =    1.28 sec.
Parallel mode: using up to 6 threads for barrier.
Number of nonzeros in lower triangle of A*A' = 1186570
Using Approximate Minimum Degree ordering
Total time for automatic ordering = 4.98 sec.
Summary statistics for Cholesky factor:
  Threads                   = 6
  Rows in Factor            = 1541
  Integer space required    = 1541
  Total non-zeros in factor = 1188111
  Total FP ops to factor    = 1220982071
 Itn      Primal Obj        Dual Obj  Prim Inf  Upper Inf  Dual Inf
   0 -1.5770000e+003 -1.6016000e+002 8.80e+002 3.15e+003 9.11e-003
   1 -7.8862128e+000 -1.6014352e+002 4.35e+000 1.56e+001 4.51e-005
   2 -8.0026282e-002 -1.5685495e+002 2.43e-016 9.43e-016 2.65e-010
   3 -8.0121260e-002 -9.5693214e-001 2.06e-018 4.88e-018 7.36e-011
   4 -1.0021787e-001 -1.6765938e-001 1.33e-017 5.43e-018 9.31e-013
   5 -1.5957418e-001 -1.6182744e-001 2.87e-017 5.35e-018 2.32e-013
   6 -1.5993991e-001 -1.5995173e-001 3.65e-018 5.38e-018 1.48e-013
   7 -1.5994210e-001 -1.5994217e-001 4.30e-018 5.43e-018 1.43e-013
   8 -1.5994212e-001 -1.5994212e-001 3.14e-018 5.37e-018 1.76e-013
   9 -1.5994212e-001 -1.5994212e-001 4.44e-018 5.48e-018 1.46e-013
```

```
43   10  -1.5994212e-001  -1.5994212e-001  1.88e-017  7.35e-018  1.98e-013
44
45 Total real time on 6 threads =   25.52 sec.
46
47 Barrier - Optimal:  Objective = -1.5994211597e-001
48 Solution time =   25.52 sec.  Iterations = 10
49
50 CPLEX> write
51 Name of file to write: QP.sol
52 Solution written to file 'QP.sol'.
```

First, we tighten the **convergence tolerance** to the minimum of 10^{-12} in line 1 to compute the solution as accurately as possible. Afterwards, we import the QP in line 4 and use CPLEX to solve it in line 8. Finally, we save the solution file in the SOL file format (5.4) in line 50.

LIBSVM: As we recorded our data sets in the LIBSVM data format like suggested in (5.4), we are able to use the software LIBSVM to compute the **prediction accuracy** (4.4.2) for a fixed parameter c for both the linear and the Gaussian kernel (4.2.5) as follows:

```
1 C:\LIBSVM_v2.91>svm-train.exe -t 0 -c 0.0001 SVM_t.lin
2 *
3 optimization finished, #iter = 231
4 nu = 0.970213
5 obj = -0.044736, rho = -0.983631
6 nSV = 457, nBSV = 453
7 Total nSV = 457
8
9 C:\LIBSVM_v2.91>svm-predict.exe SVM_p.lin SVM_t.lin.model SVM_p.
    out
10 Accuracy = 52% (1131/2175) (classification)
```

In this case, LIBSVM solves the SVM defined by our training set $\mathcal{T}$ (4.2.1) in line 1, where "-t 0" initializes the **linear kernel** and "-c" sets the parameter c to the specified value. Note that LIBSVM writes the solution automatically to a "*.model" file. To compute the prediction accuracy, we use the prediction set $\mathcal{P}$ (4.4.2) in line 9 which is also given in the LIBSVM data format. Additionally, the prediction needs the already computed "*.model" file and an output file name.

To use the **Gaussian kernel** on the same problem setup, the argument "-t 2" replaces "-t 0". Furthermore, it is possible to specify the parameter γ with the "-g" argument which is set to the following default value,

$$\gamma_{def} := \frac{1}{\mathrm{n}},$$

by LIBSVM, where $x^{(i)} \in \mathbb{R}^n$, $i \in \{1, \ldots, m\}$ (4.2.1).

Note that we also used γ_{def} for the **Gaussian kernel** in the following tests (5.6) of our implementation of the criss-cross method (4.1).

5.6 Experimental Results

5.6.1 Introduction

As we tested our implementation of the criss-cross method (4.1), we collected all the **experimental results** presented in this subsection. While using the linear and the Gaussian kernel for each data set as typical kernel representatives (4.2.5), we recorded the following:

- **running time**
- **regularization parameter** c and **path length** (4.1.4)
- **value** $|B_0^*|$ (4.4.4)

  According to the experimental results, $|B_0^*| \approx 2\,|l - k| + 2$ holds for all tested data sets. The start bases were determined as described in (4.4.1) for $|\mathcal{T}_+| \neq |\mathcal{T}_-|$ (4.2.1), whereas the artificial start basis B_{art} (4.4.3) was used for the CeBIT2 data set where $|\mathcal{T}_+| = |\mathcal{T}_-|$ holds.

- **prediction accuracy** (4.4.2)

  As is shown in the diagrams, the linear kernel provides the best results in general, whereas the Gaussian kernel causes overfitting for a growing regularization parameter c, see for example the CeBIT1 data set (5.6.5). This proposition is true except for the Splice data set (5.6.11), where the Gaussian kernel causes no overfitting but a significantly better prediction accuracy.

- **pattern vectors' locations** (4.6)

  The geometrical difference between the linear and the Gaussian kernel is very well visible in all diagrams: The number of vectors on the margin (4.2.1) remains nearly the same in the linear case, whereas it increases significantly while using the Gaussian kernel creating a non-linear separation, see for example the Adult data set (5.6.9).

- **value** b [(29), p.21)]

  We recorded the characteristics of the value b to demonstrate the specialty of a conjoint analysis problem (4.5.3), where $|\mathcal{T}_+| = |\mathcal{T}_-|$ is generated by adding redundant information. As described in [(53), p.39], $b = 0$ holds in this case as one can see in the CeBIT2 data set diagram (5.6.6) and (5.6.7).

Additionally, we compared our prediction results with LIBSVM (4.4.2) using 141 random samples resulting in a small difference of 4.15% in our favor. The following two types of data sets contain both a training set $\mathcal{T}$ and a prediction set $\mathcal{P}$ (4.4.2).

Conjoint Data Sets: All data sets in this category were surveyed under the leadership of Prof. Dr. Joachim Giesen at the University of Jena. Additionally to the records mentioned above, we computed the **part worth values** (4.5.4) along the regularization path while using the linear kernel where changes up to 23% occured, see for example the Engine data set (5.6.2). In the following, the data sets are described in detail:

Engine: $|\mathcal{T}| = 1382$, $|\mathcal{P}| = 6220$, $n = 24$, $|\mathcal{T}_+| = 685$, $|\mathcal{T}_-| = 697$

For a detailed description of the data set and the survey, feel free to contact the chair in Theoretical Computer Science II under the leadership of Prof. Dr. Joachim Giesen at the University of Jena.

CeBIT1: $|\mathcal{T}| = 900$, $|\mathcal{P}| = 408$, $n = 20$, $|\mathcal{T}_+| = 435$, $|\mathcal{T}_-| = 465$

For a detailed description of the data set and the survey, feel free to contact the chair in Theoretical Computer Science II under the leadership of Prof. Dr. Joachim Giesen at the University of Jena.

CeBIT2: $|\mathcal{T}| = 1800$, $|\mathcal{P}| = 408$, $n = 20$, $|\mathcal{T}_+| = 900$, $|\mathcal{T}_-| = 900$

This data set is based on CeBIT1 and was created as described in (4.5.3) resulting in $|\mathcal{T}_+| = |\mathcal{T}_-|$ to be able to use B_{art} as start basis (4.4.3). As one can easily see, $b = 0$ holds along the entire regularization path as described in [(53), p.39].

SVM Data Sets: The first and the second data set were taken from the LIBSVM data sets page[13], the third set was artificially created for a special purpose, see below.

Adult: $|\mathcal{T}| = 1605$, $|\mathcal{P}| = 30956$, $n = 123$, $|\mathcal{T}_+| = 395$, $|\mathcal{T}_-| = 1210$

For a detailed description, please follow the link mentioned above.

Splice: $|\mathcal{T}| = 470$, $|\mathcal{P}| = 2175$, $n = 60$, $|\mathcal{T}_+| = 228$, $|\mathcal{T}_-| = 242$

For a detailed description, please follow the link mentioned above.

Simple: $|\mathcal{T}| = 7$, $n = 2$, $|\mathcal{T}_+| = 4$, $|\mathcal{T}_-| = 3$

This little data set was created to describe the criss-cross algorithm (4.1.3) step by step during the entire regularization path. To simplify the problem setup, we used no prediction set $\mathcal{P}$ and only the linear kernel in this case. Note that the data set is visualized in (4.2.2) with $c = 2$.

[13] http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/ (25.06.2010).

5.6.2 Engine Linear Kernel

$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{P}\rvert$	n	running time	path length	$\lvert B_0^*\rvert$	$\lvert\mathcal{T}_+\rvert$	$\lvert\mathcal{T}_-\rvert$
1617	7287	24	140min	1850	213	861	756

Table 3: Engine data set overview using the linear kernel.

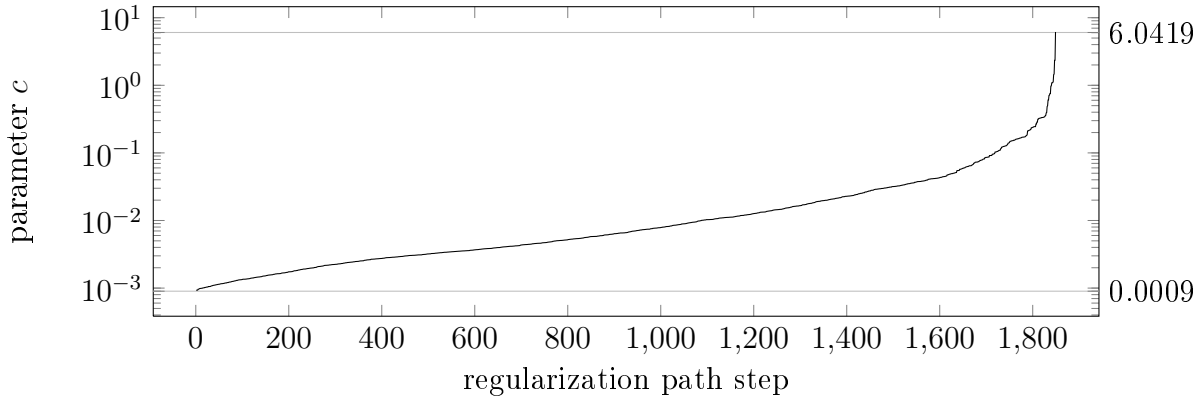

Figure 14: Engine | Linear Kernel | Regularization Parameter

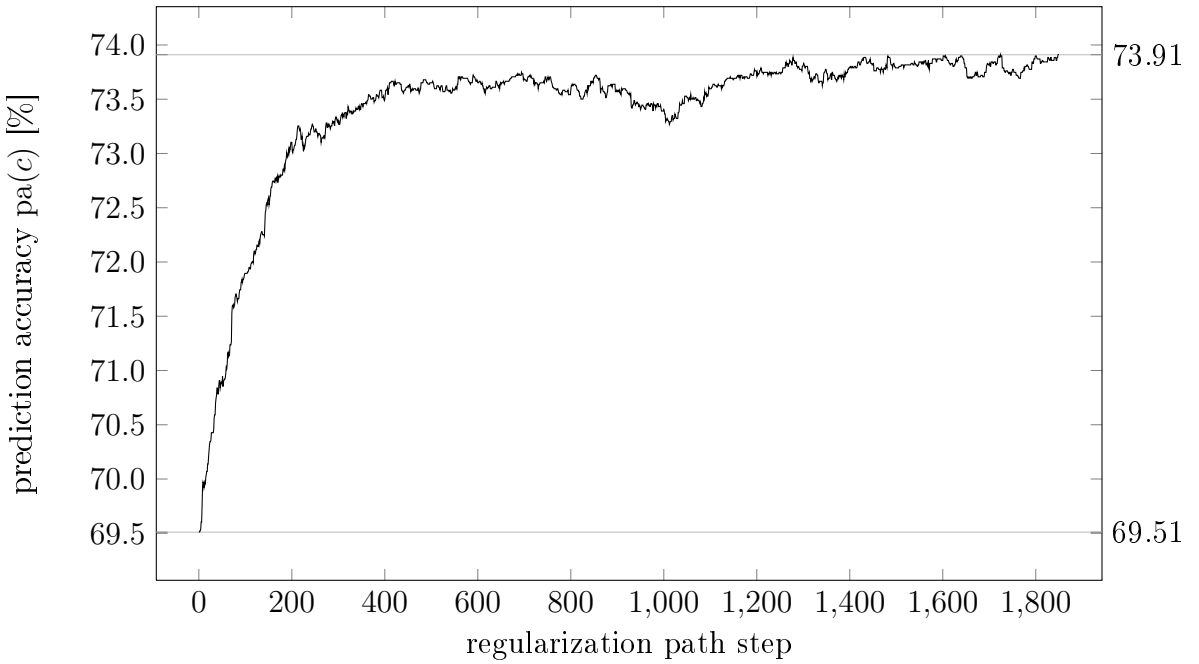

Figure 15: Engine | Linear Kernel | Prediction Accuracy

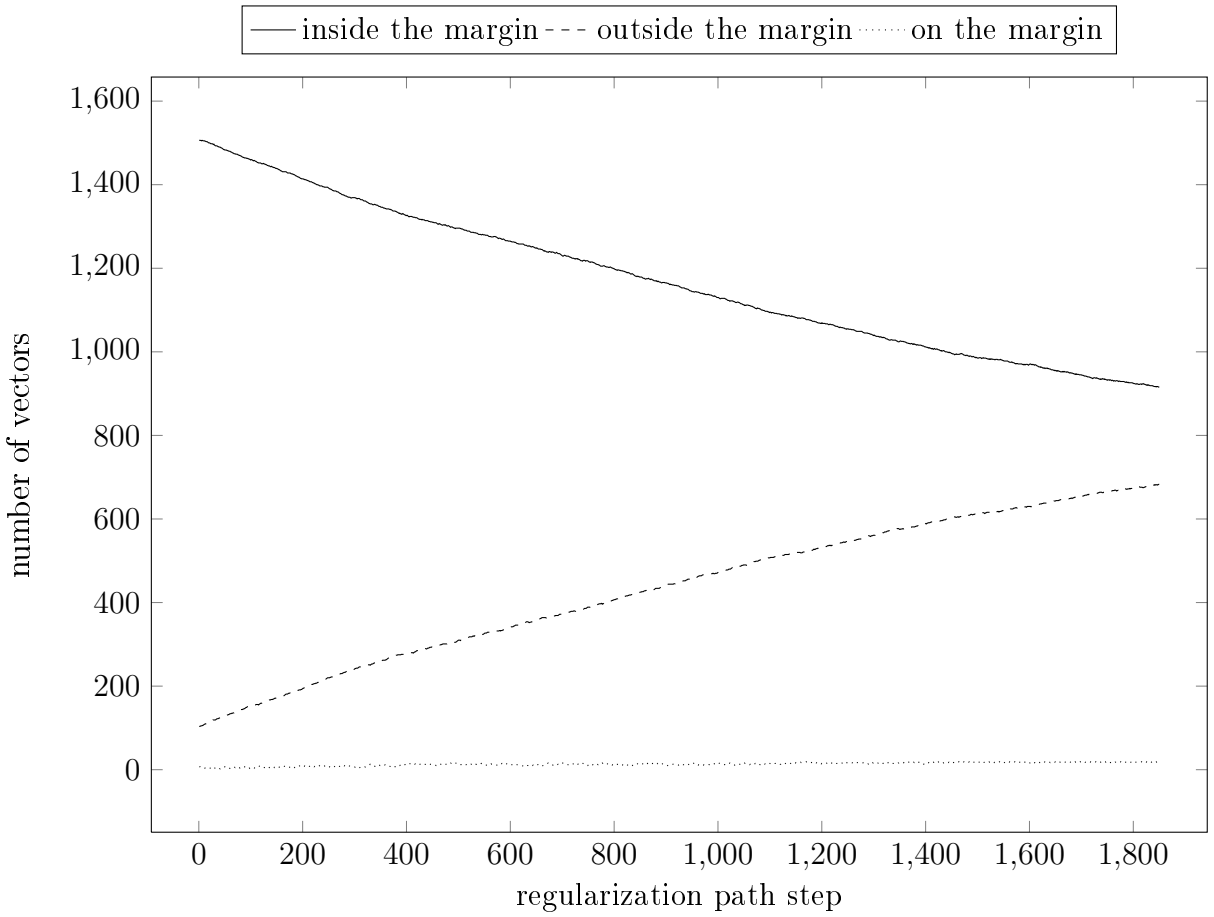

Figure 16: Engine | Linear Kernel | Vectors' Locations

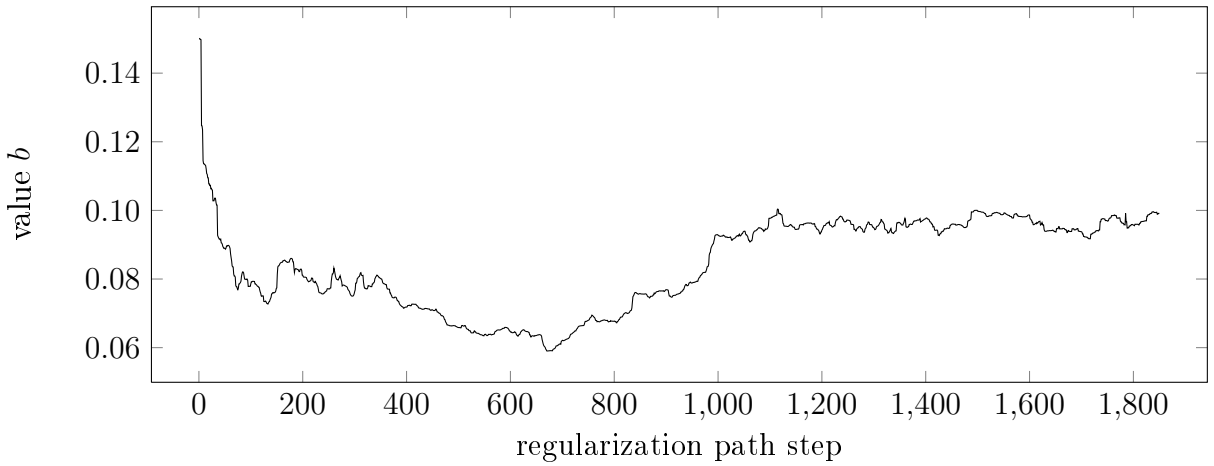

Figure 17: Engine | Linear Kernel | Value b

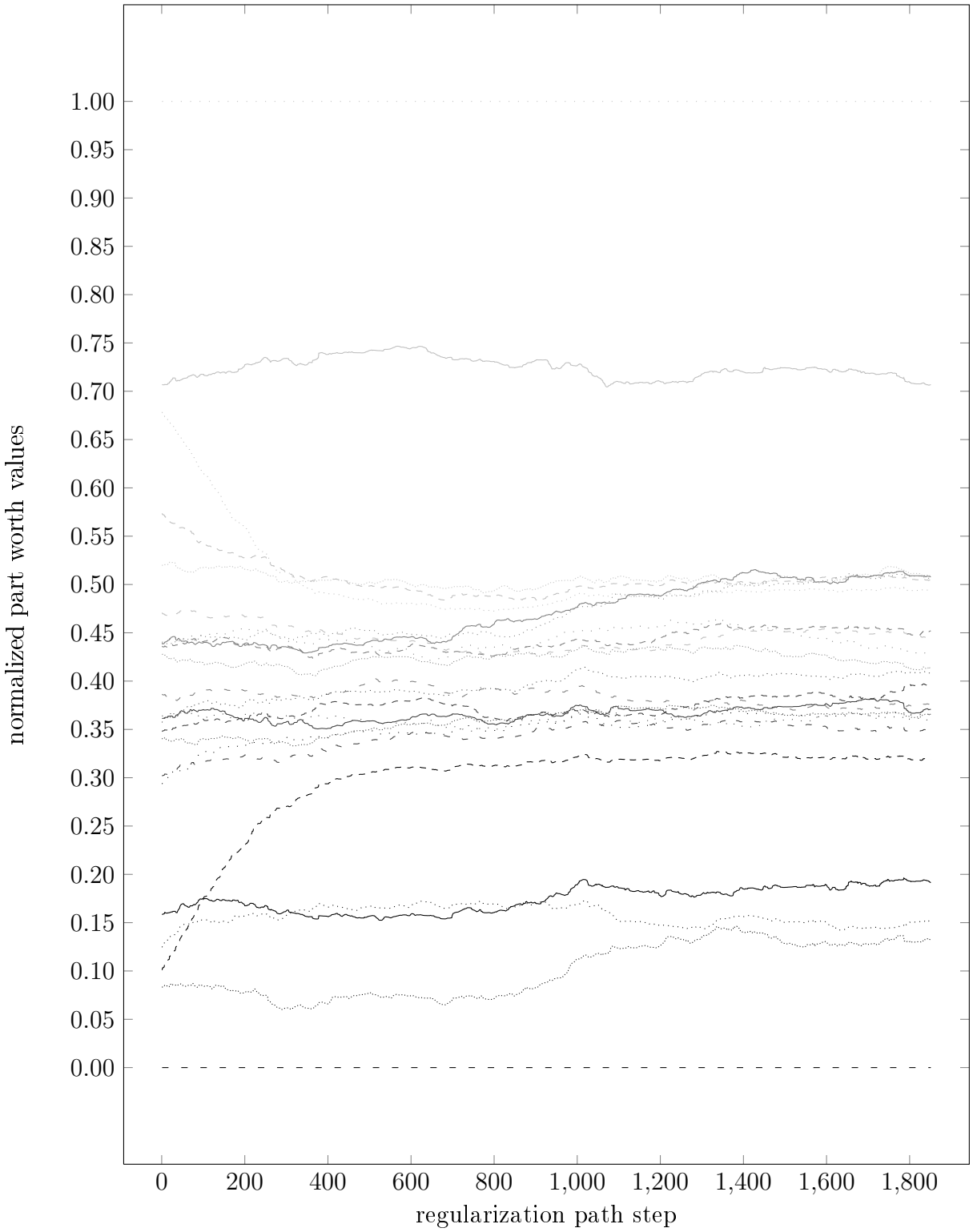

Figure 18: Engine | Linear Kernel | Part Worth Values

5.6.3 Engine Gaussian Kernel

$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{P}\rvert$	n	running time	path length	$\lvert B_0^*\rvert$	$\lvert\mathcal{T}_+\rvert$	$\lvert\mathcal{T}_-\rvert$
1617	7287	24	13h	3202	213	435	465

Table 4: Engine data set overview using the Gaussian kernel.

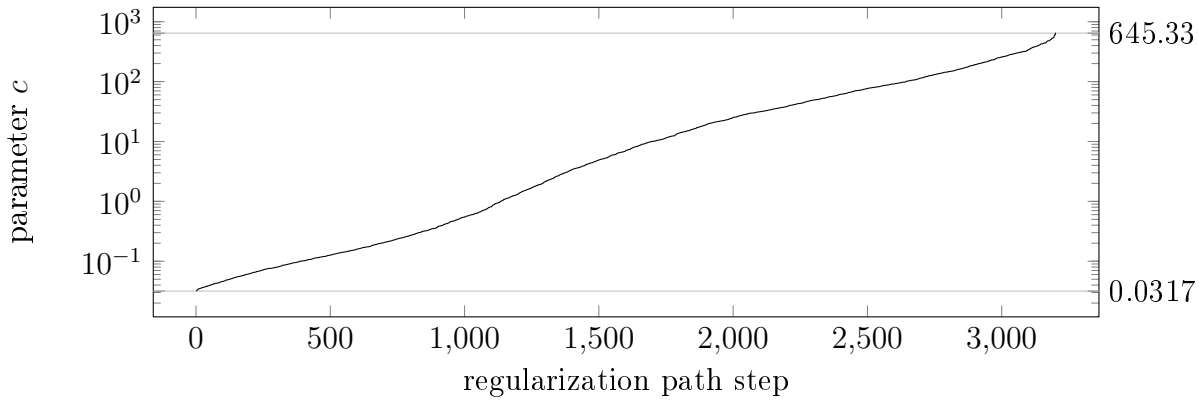

Figure 19: Engine | Gaussian Kernel | Regularization Parameter

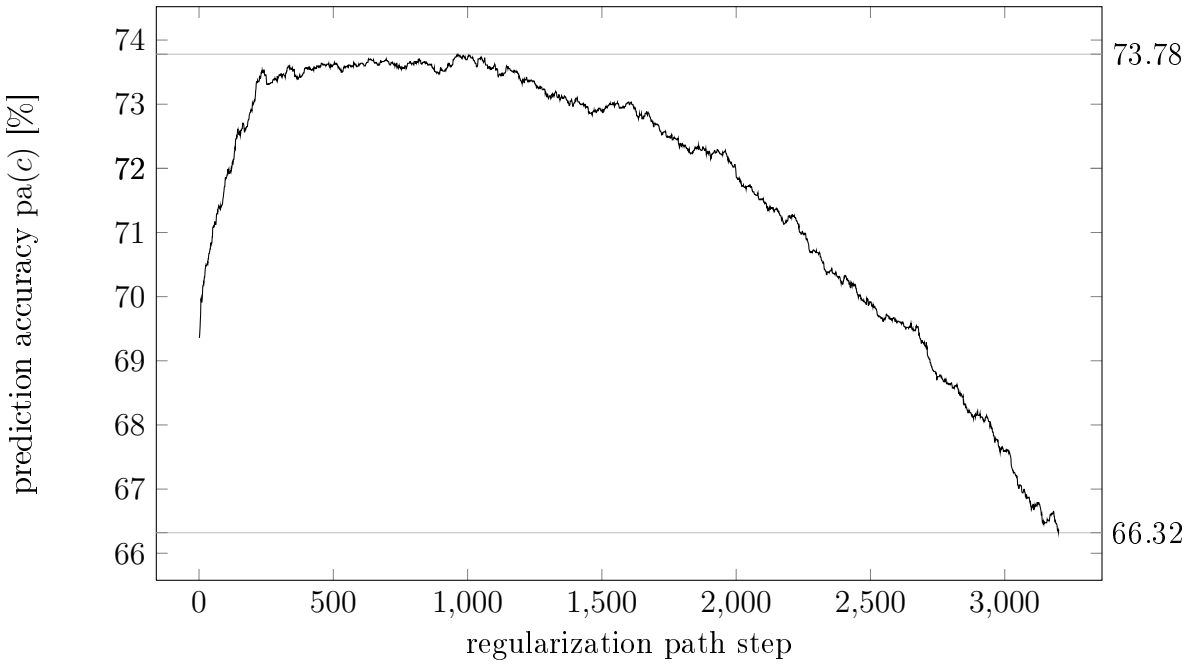

Figure 20: Engine | Gaussian Kernel | Prediction Accuracy

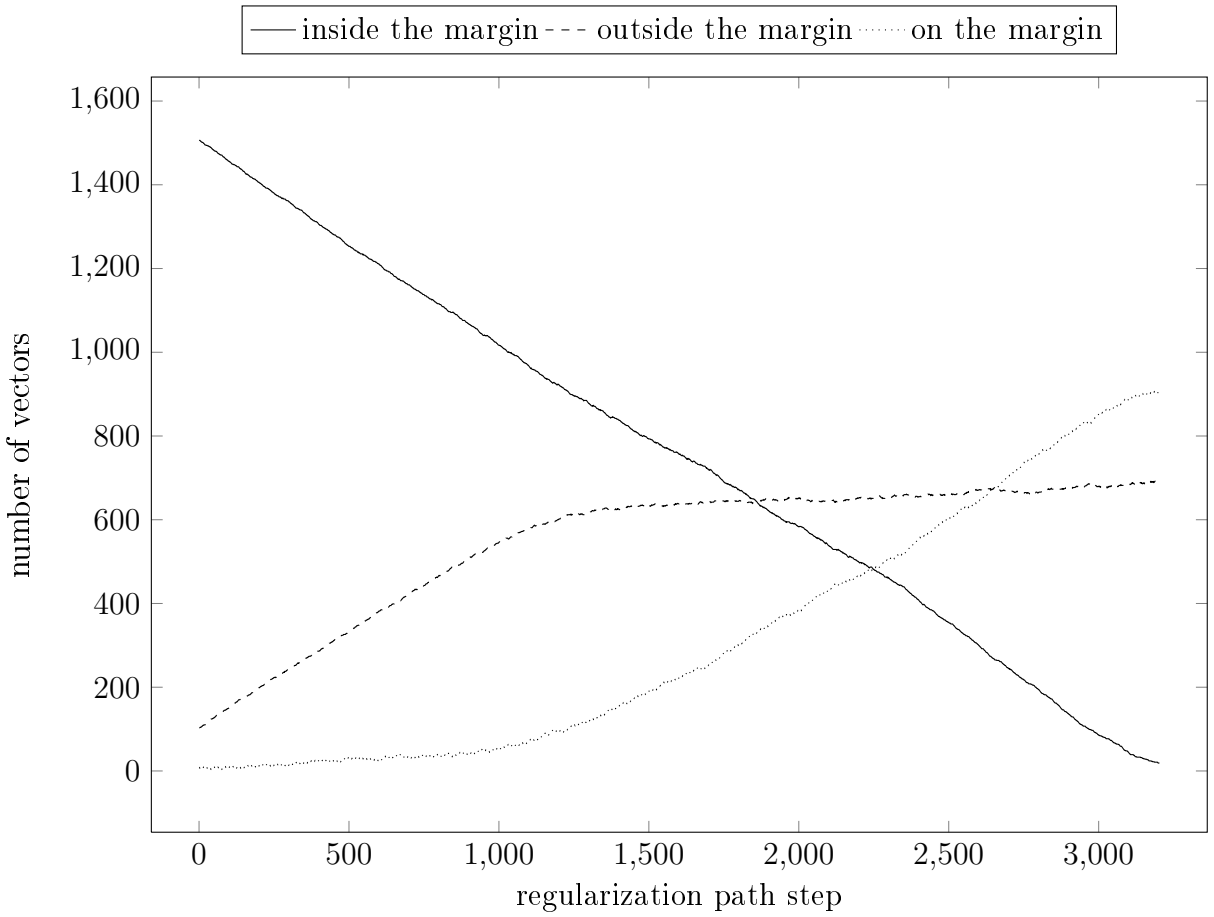

Figure 21: Engine | Gaussian Kernel | Vectors' Locations

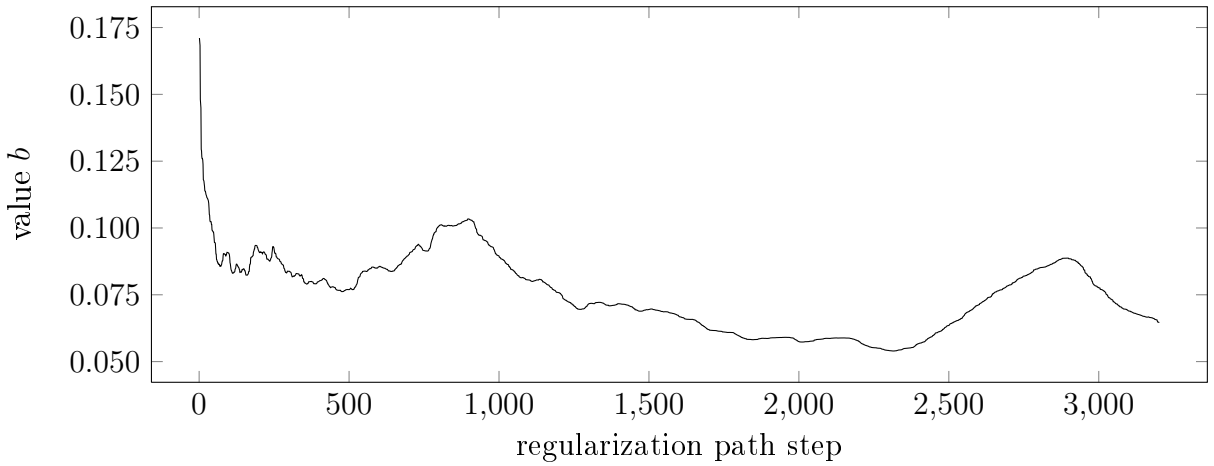

Figure 22: Engine | Gaussian Kernel | Value b

5.6.4 CeBIT1 Linear Kernel

$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{P}\rvert$	n	running time	path length	$\lvert B_0^*\rvert$	$\lvert\mathcal{T}_+\rvert$	$\lvert\mathcal{T}_-\rvert$
900	408	20	14min	832	62	435	465

Table 5: CeBIT1 data set overview using the linear kernel.

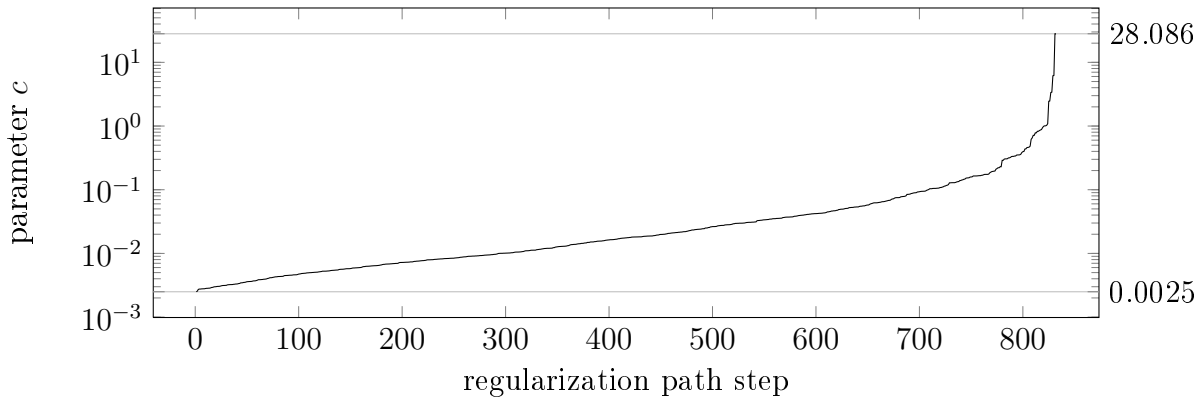

Figure 23: CeBIT1 | Linear Kernel | Regularization Parameter

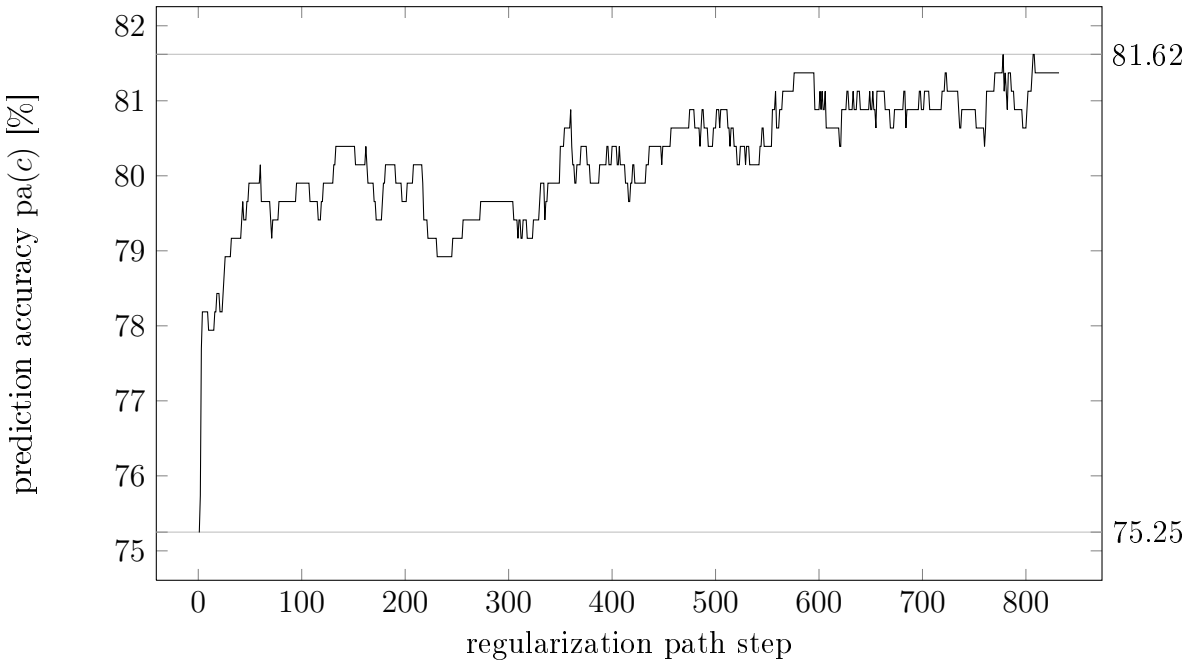

Figure 24: CeBIT1 | Linear Kernel | Prediction Accuracy

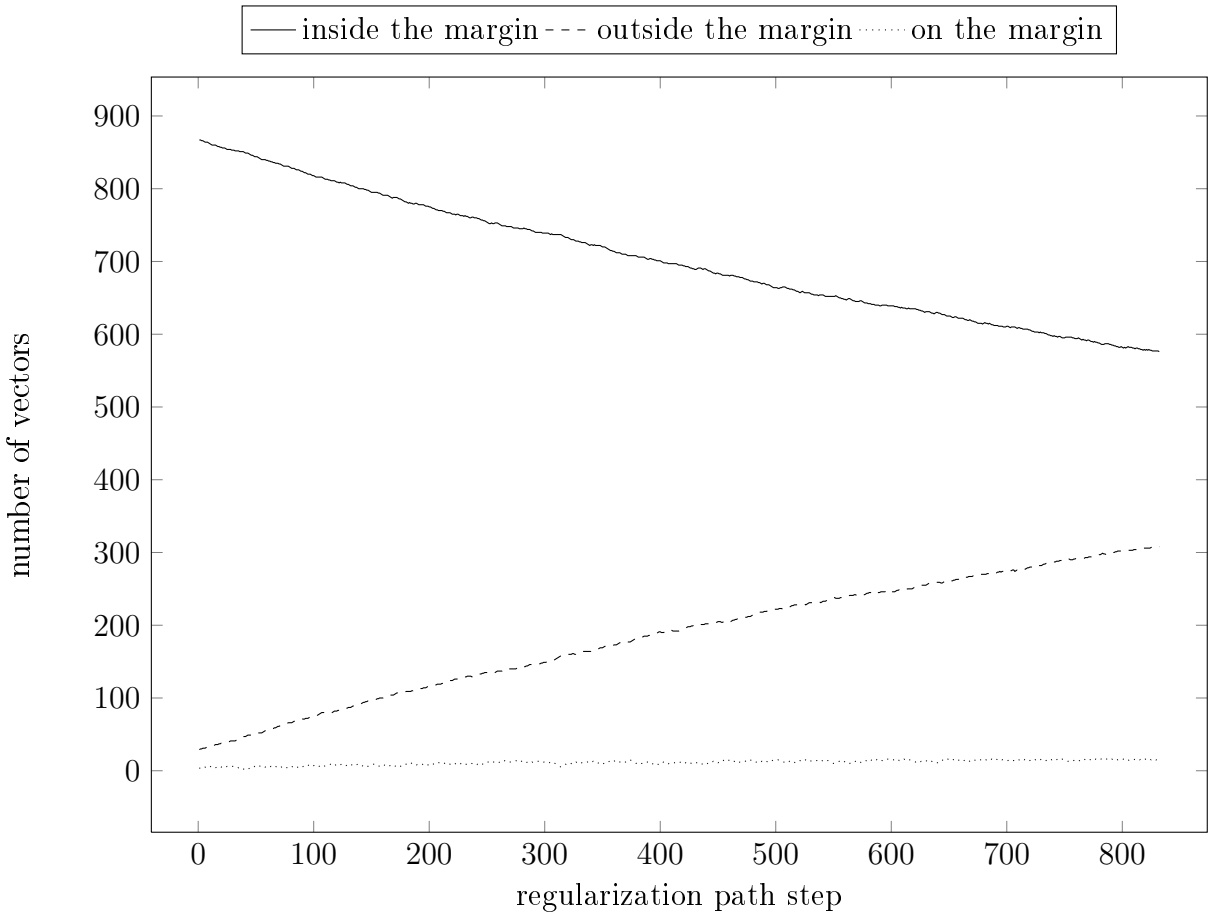

Figure 25: CeBIT1 | Linear Kernel | Vectors' Locations

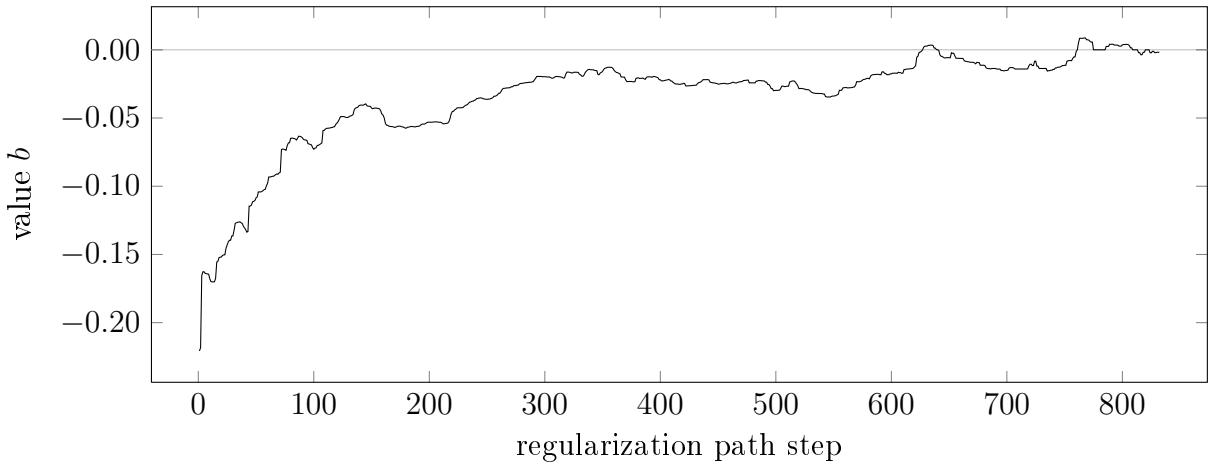

Figure 26: CeBIT1 | Linear Kernel | Value b

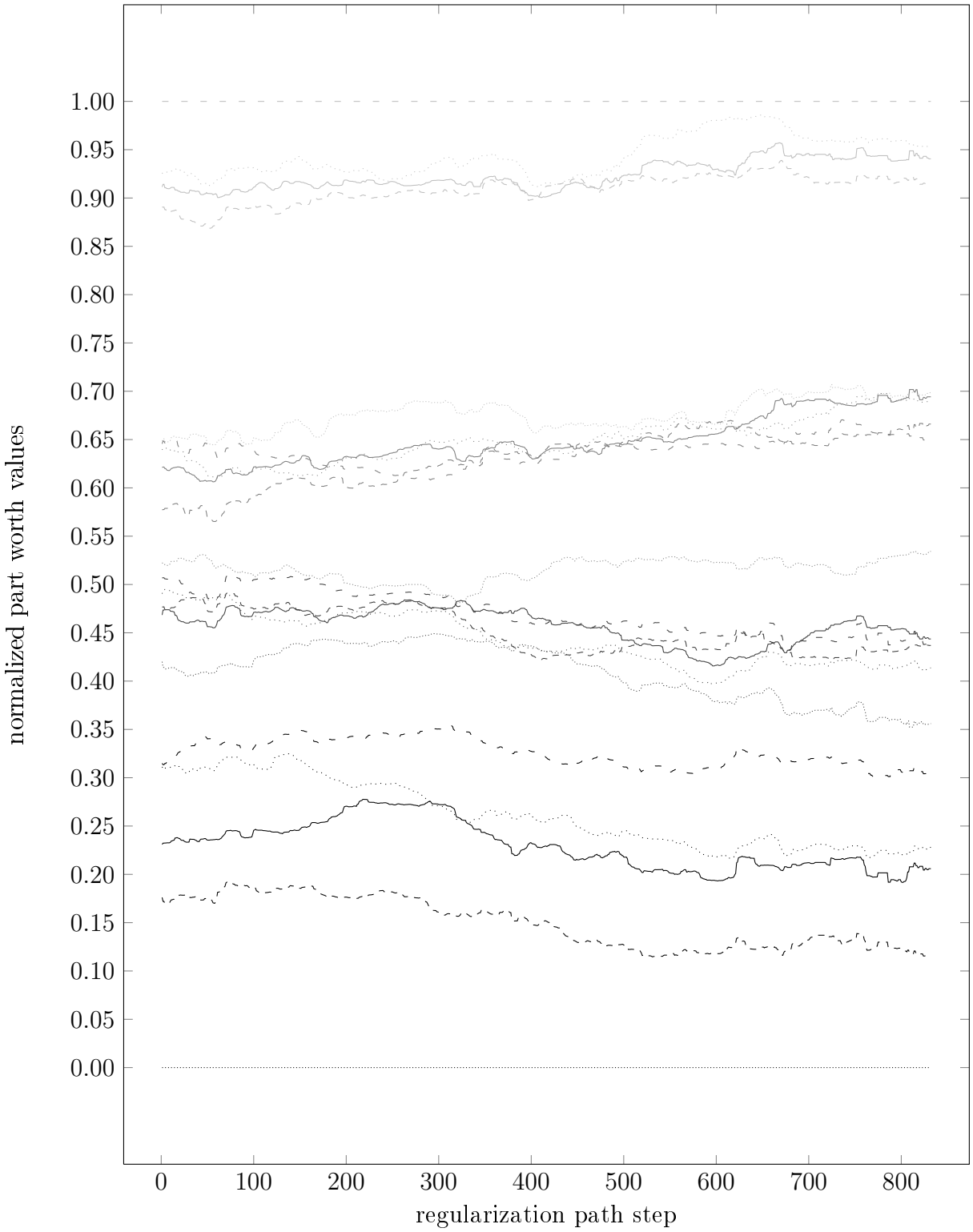

Figure 27: CeBIT1 | Linear Kernel | Part Worth Values

5.6.5 CeBIT1 Gaussian Kernel

$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{P}\rvert$	n	running time	path length	$\lvert B_0^*\rvert$	$\lvert\mathcal{T}_+\rvert$	$\lvert\mathcal{T}_-\rvert$
900	408	20	108min	1860	60	435	465

Table 6: CeBIT1 data set overview using the Gaussian kernel.

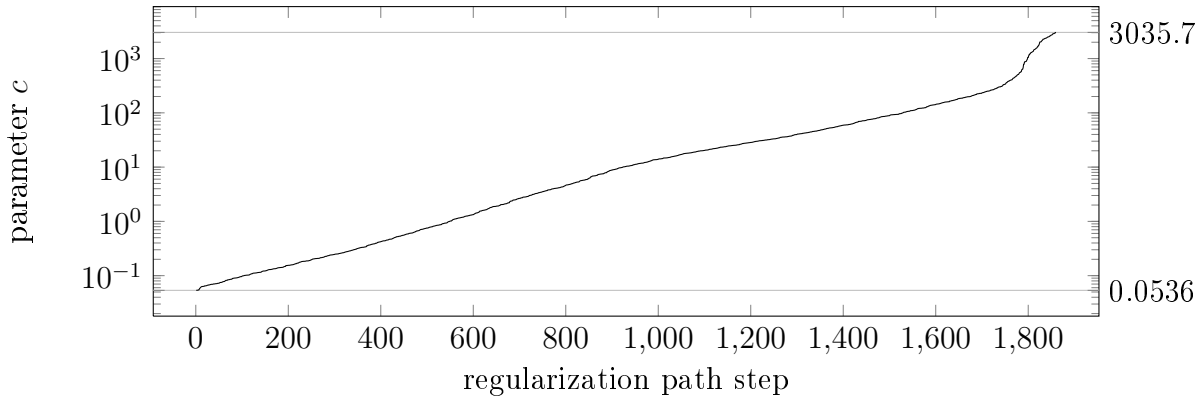

Figure 28: CeBIT1 | Gaussian Kernel | Regularization Parameter

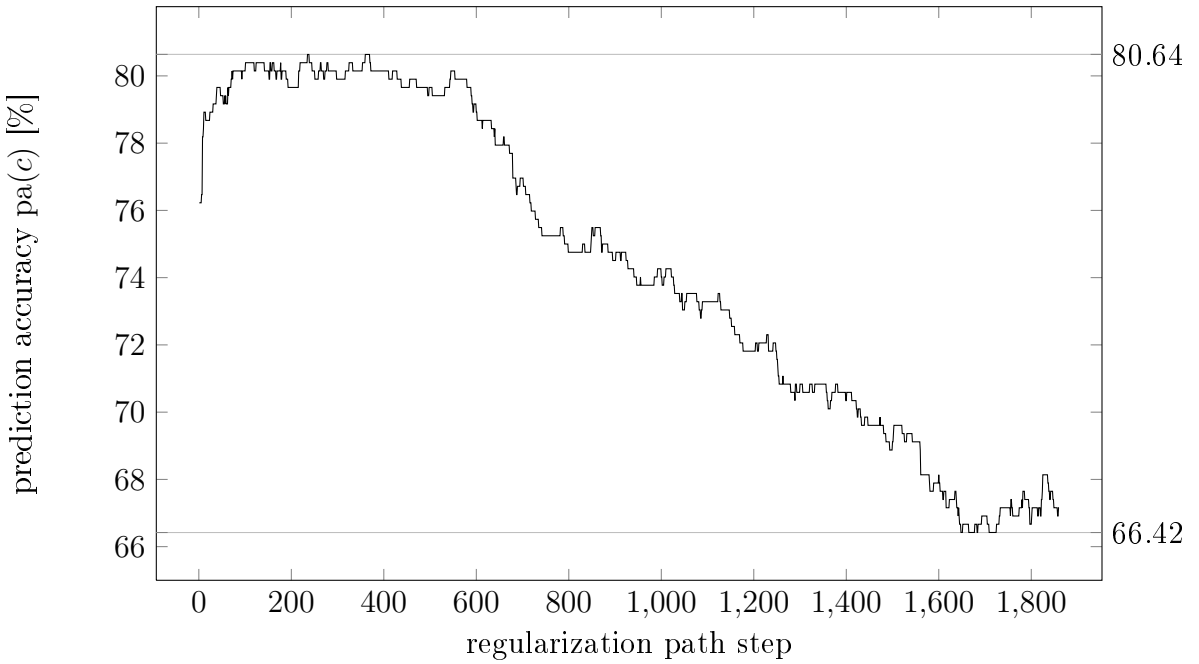

Figure 29: CeBIT1 | Gaussian Kernel | Prediction Accuracy

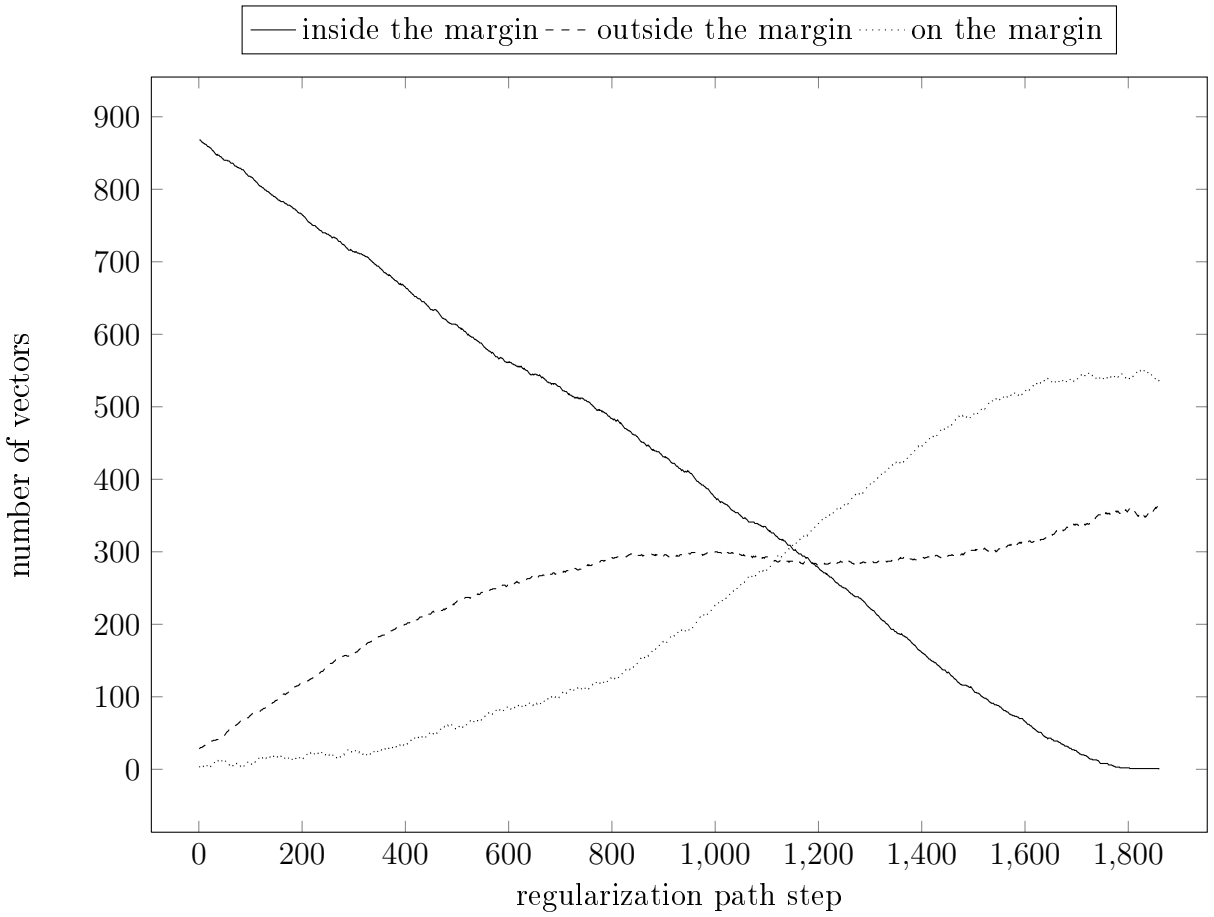

Figure 30: CeBIT1 | Gaussian Kernel | Vectors' Locations

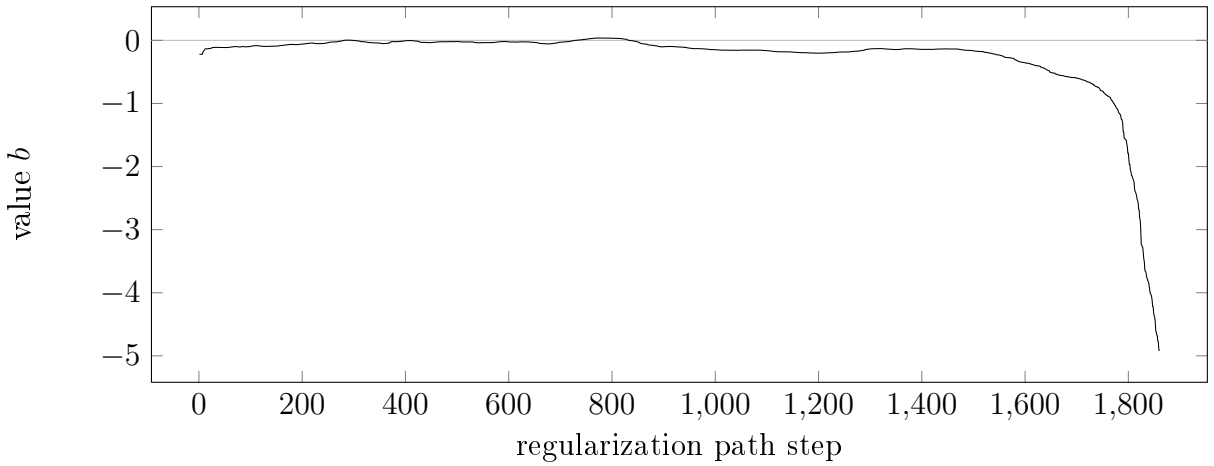

Figure 31: CeBIT1 | Gaussian Kernel | Value b

5.6.6 CeBIT2 Linear Kernel

$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{P}\rvert$	n	running time	path length	$\lvert B_0^*\rvert$	$\lvert\mathcal{T}_+\rvert$	$\lvert\mathcal{T}_-\rvert$
1800	408	20	122min	1744	2	900	900

Table 7: CeBIT2 data set overview using the linear kernel.

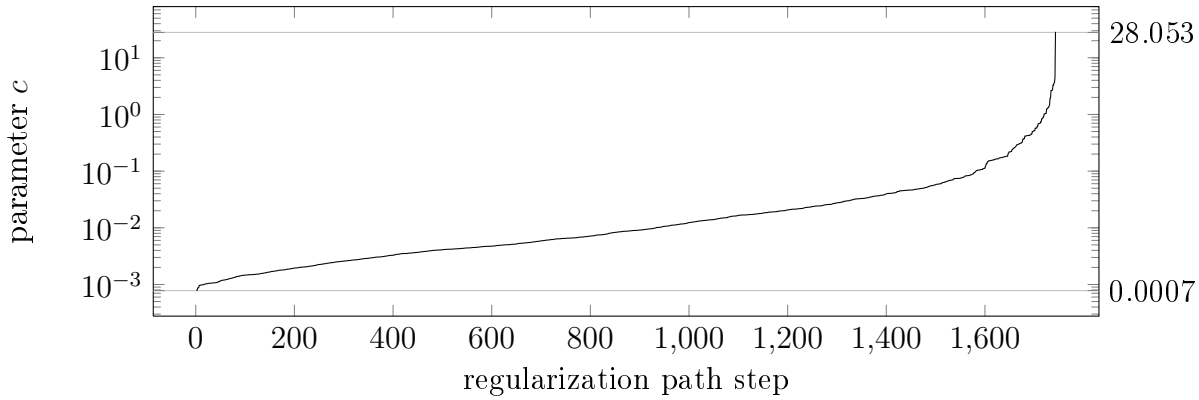

Figure 32: CeBIT2 | Linear Kernel | Regularization Parameter

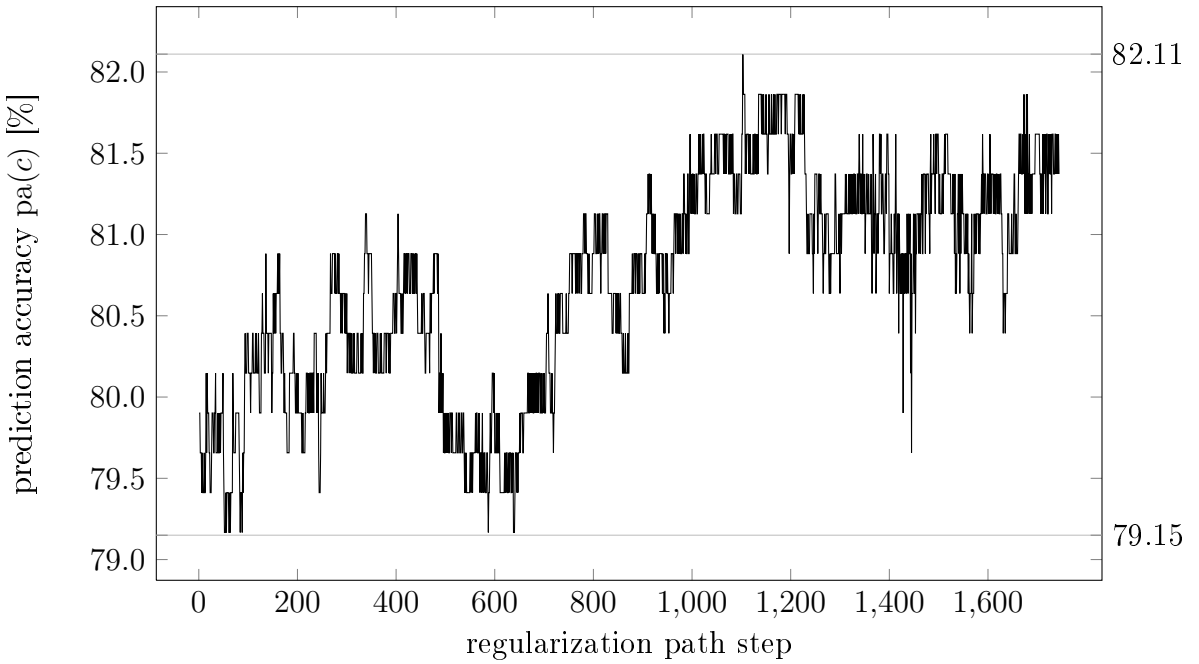

Figure 33: CeBIT2 | Linear Kernel | Prediction Accuracy

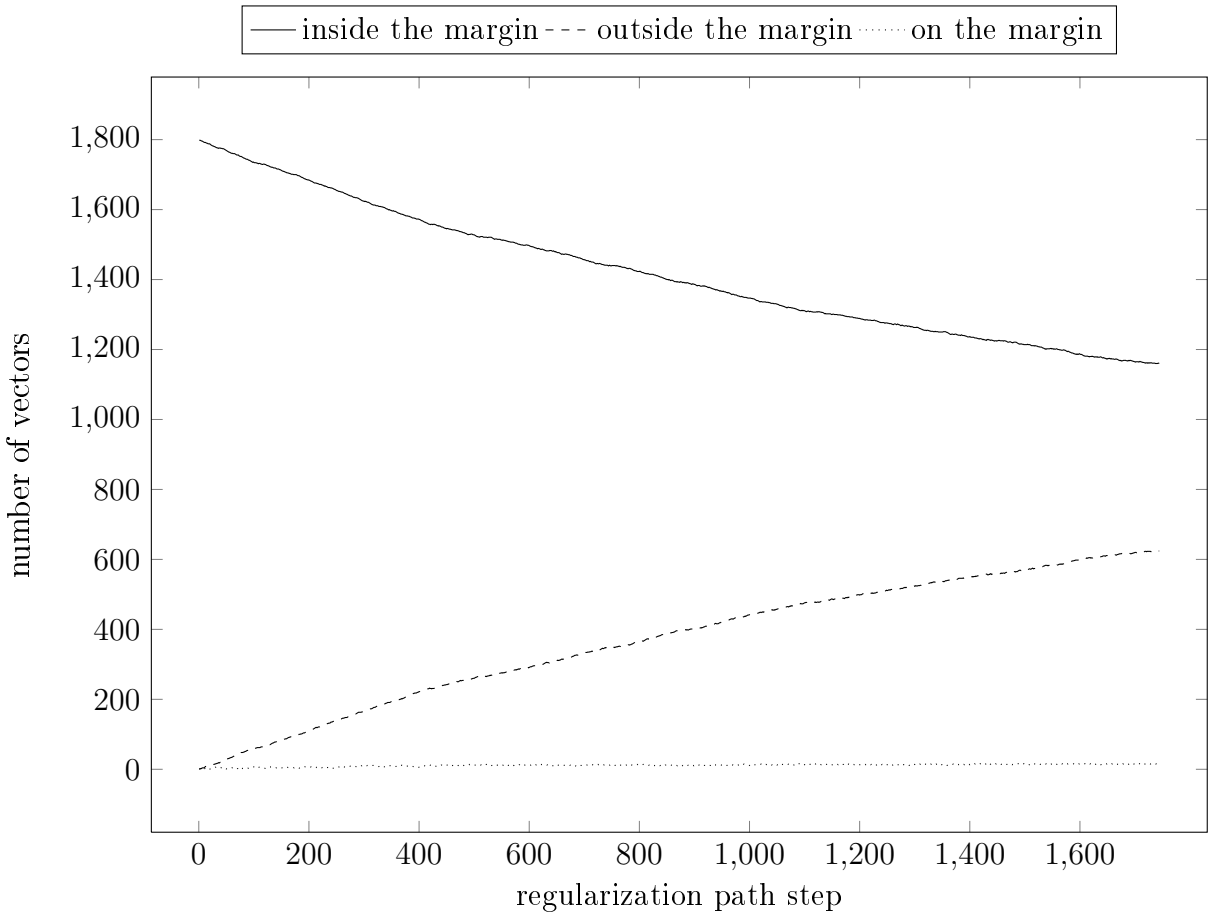

Figure 34: CeBIT2 | Linear Kernel | Vectors' Locations

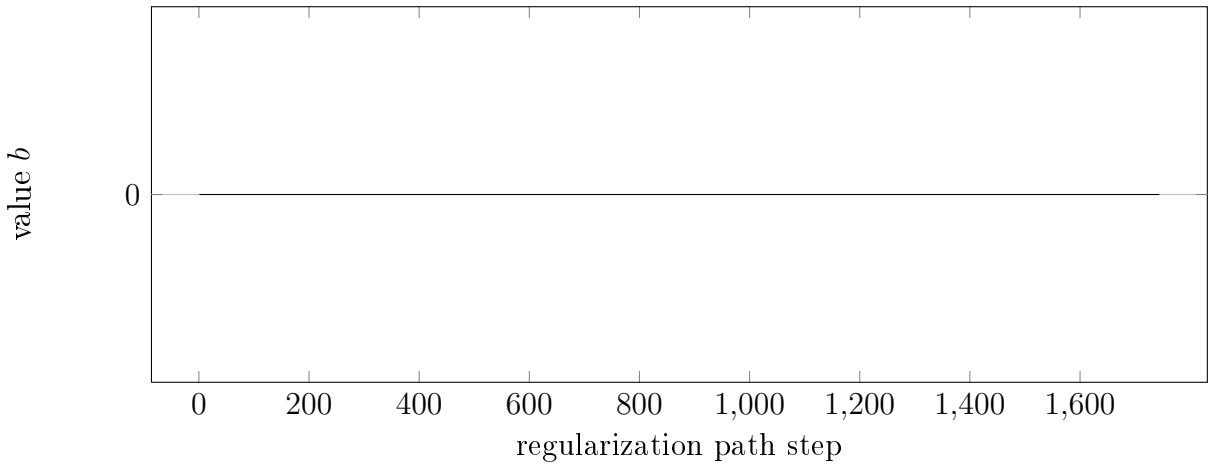

Figure 35: CeBIT2 | Linear Kernel | Value b

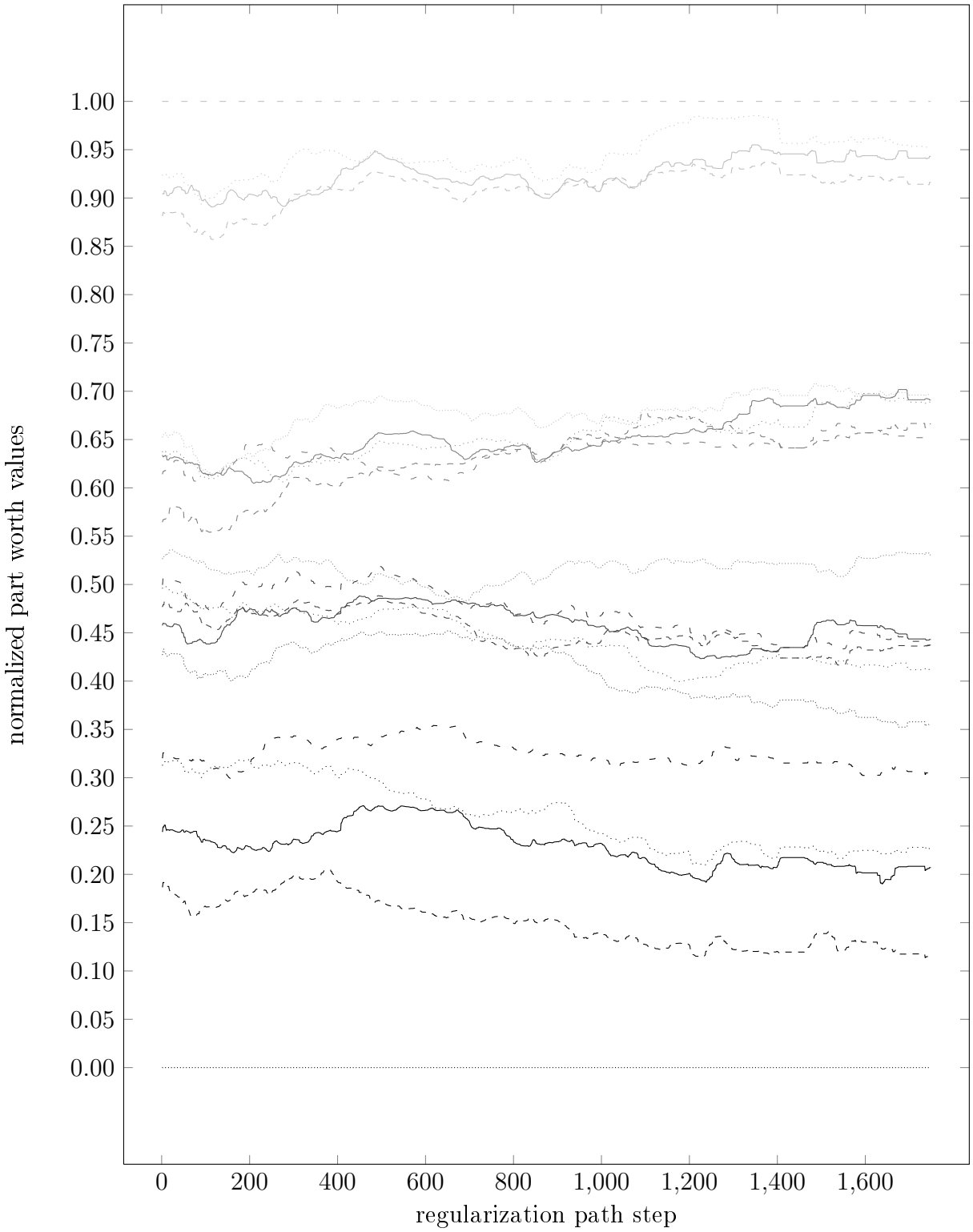

Figure 36: CeBIT2 | Linear Kernel | Part Worth Values

5.6.7 CeBIT2 Gaussian Kernel

$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{P}\rvert$	n	running time	path length	$\lvert B_0^*\rvert$	$\lvert\mathcal{T}_+\rvert$	$\lvert\mathcal{T}_-\rvert$
1800	408	20	21h	3165	2	900	900

Table 8: CeBIT2 data set overview using the Gaussian kernel.

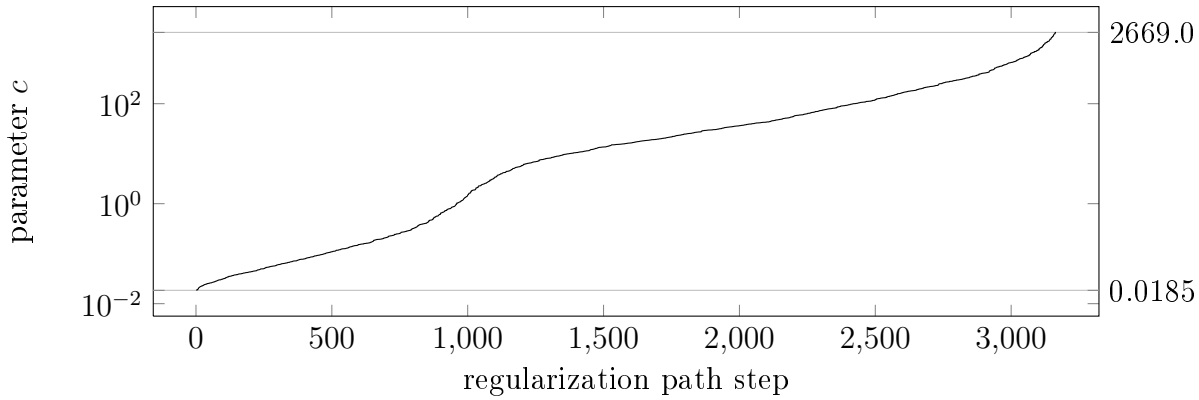

Figure 37: CeBIT2 | Gaussian Kernel | Regularization Parameter

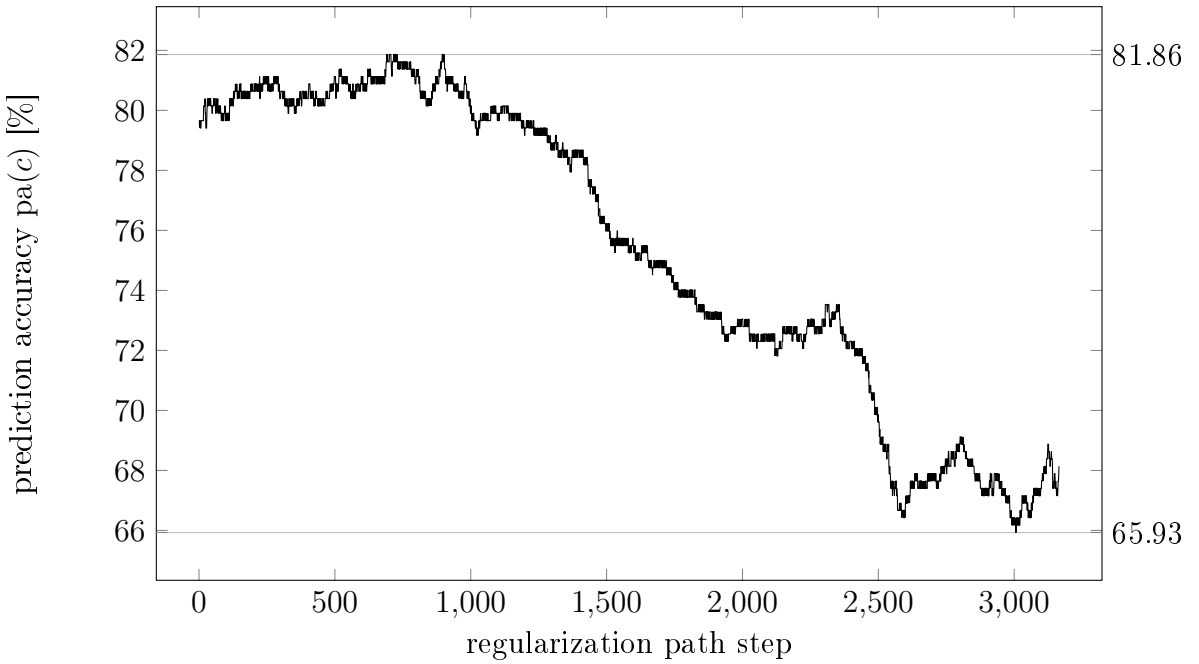

Figure 38: CeBIT2 | Gaussian Kernel | Prediction Accuracy

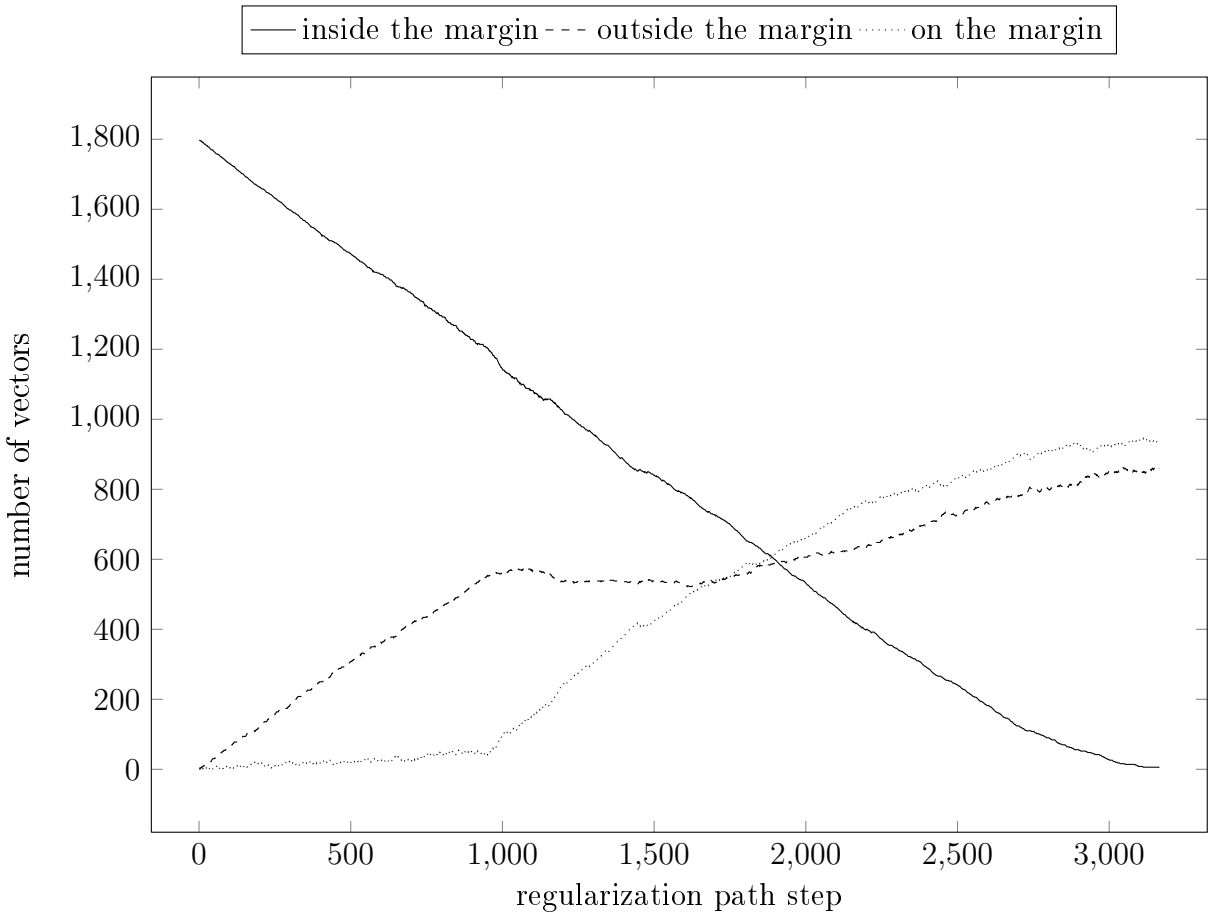

Figure 39: CeBIT2 | Gaussian Kernel | Vectors' Locations

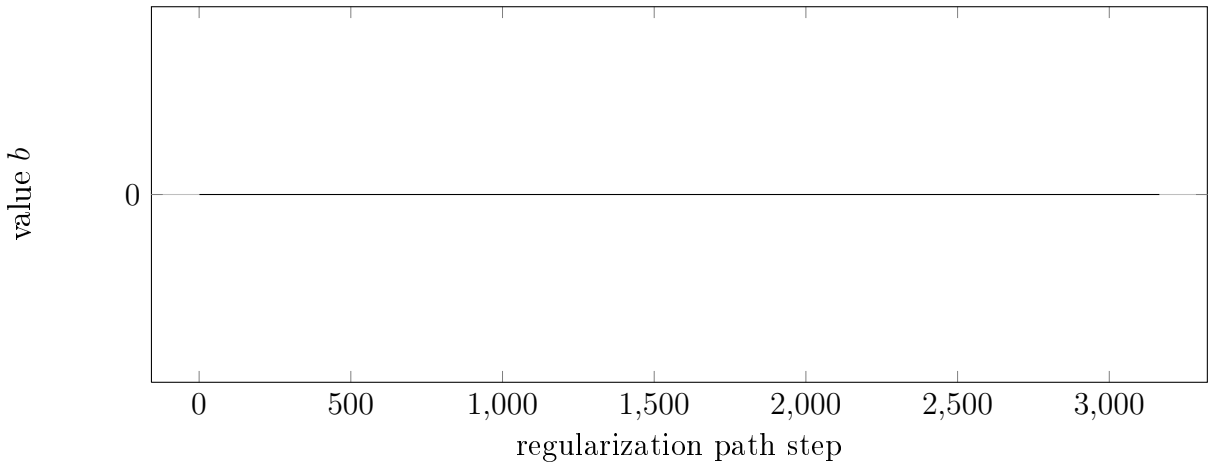

Figure 40: CeBIT2 | Gaussian Kernel | Value b

5.6.8 Adult Linear Kernel

$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{P}\rvert$	n	running time	path length	$\lvert B_0^*\rvert$	$\lvert\mathcal{T}_+\rvert$	$\lvert\mathcal{T}_-\rvert$
1605	30956	123	12h	1460	1628	395	1210

Table 9: Adult data set overview using the linear kernel.

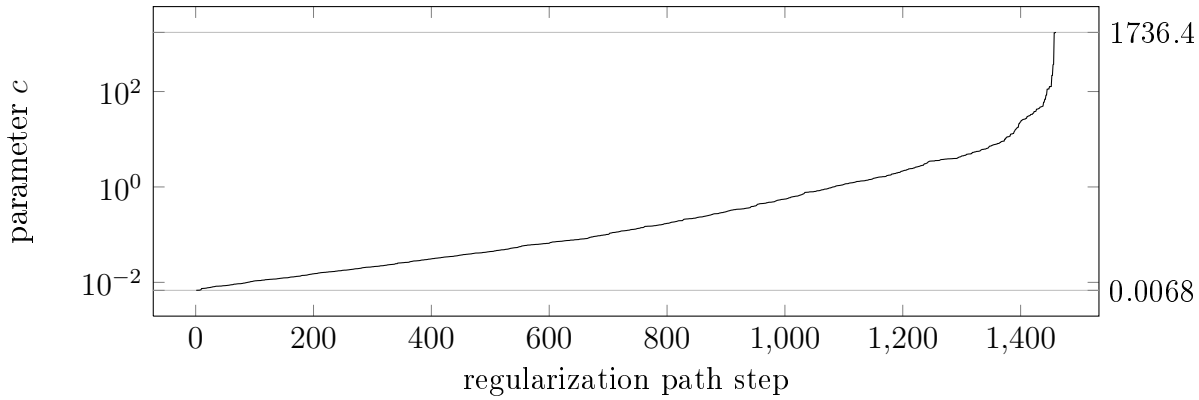

Figure 41: Adult | Linear Kernel | Regularization Parameter

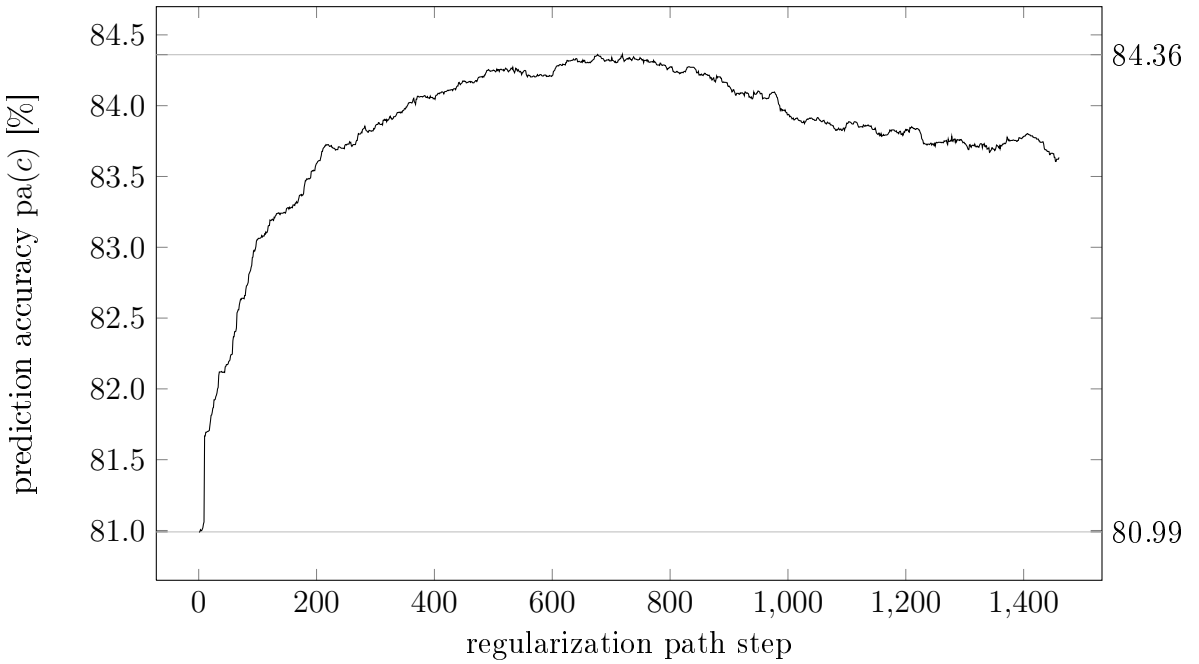

Figure 42: Adult | Linear Kernel | Prediction Accuracy

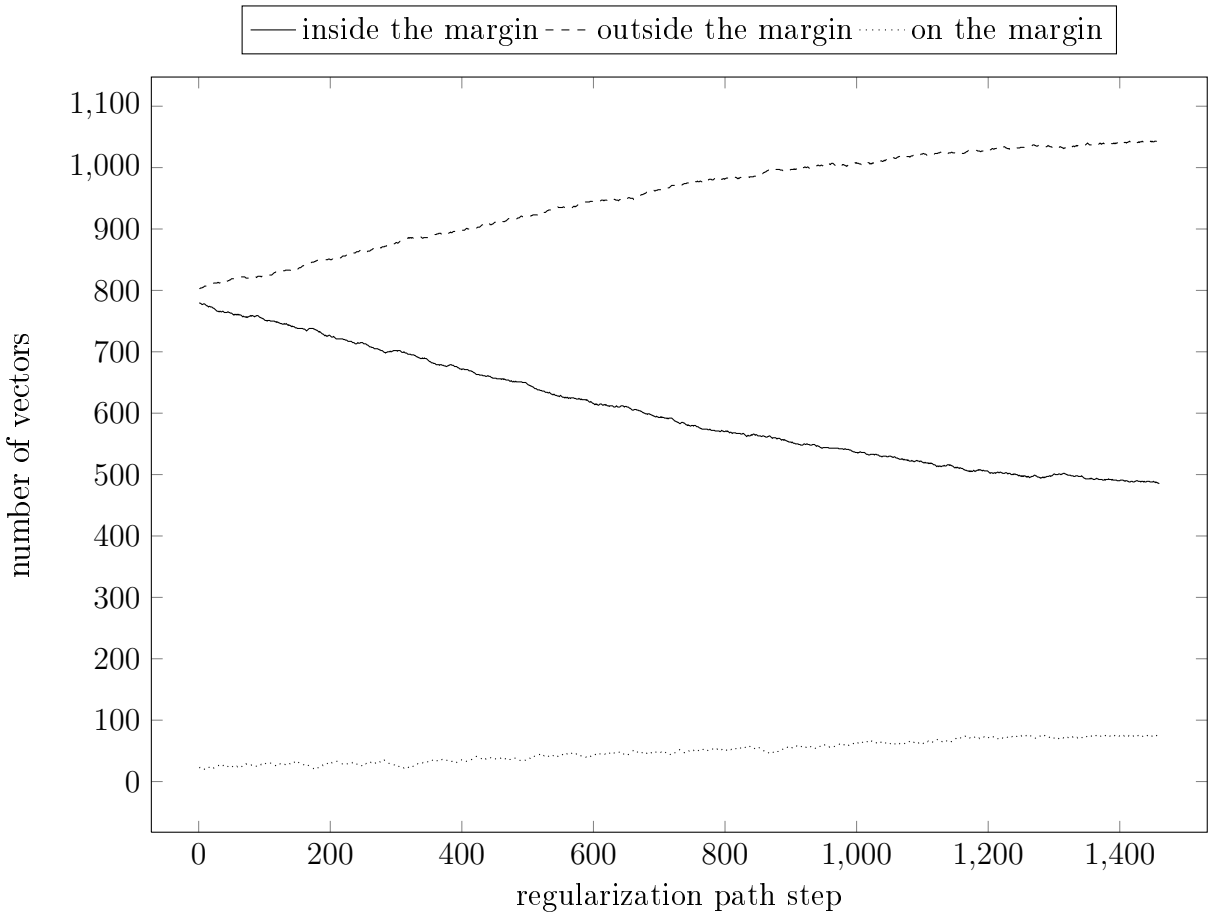

Figure 43: Adult | Linear Kernel | Vectors' Locations

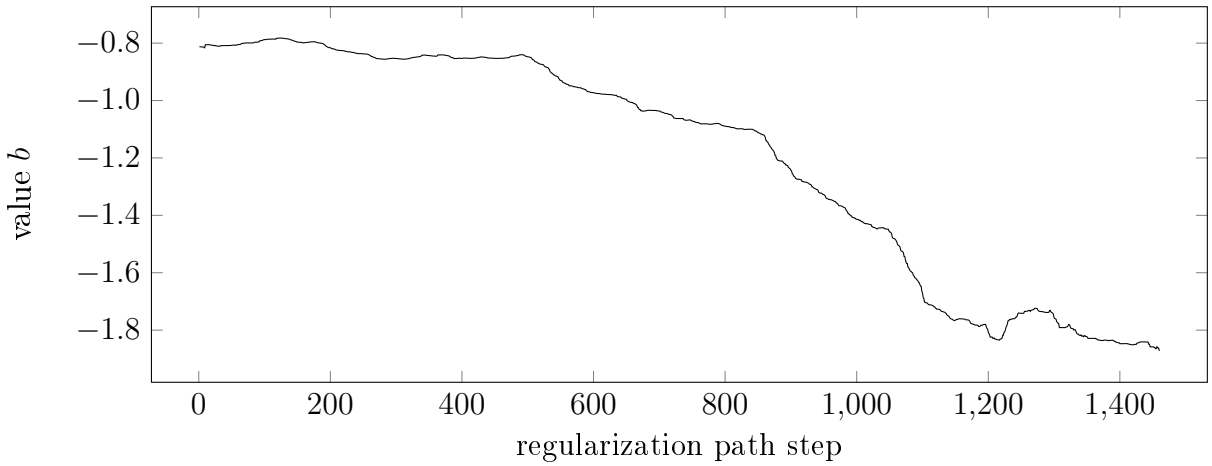

Figure 44: Adult | Linear Kernel | Value b

5.6.9 Adult Gaussian Kernel

$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{P}\rvert$	n	running time	path length	$\lvert B_0^*\rvert$	$\lvert\mathcal{T}_+\rvert$	$\lvert\mathcal{T}_-\rvert$
1605	30956	123	22h	1585	1639	395	1210

Table 10: Adult data set overview using the Gaussian kernel.

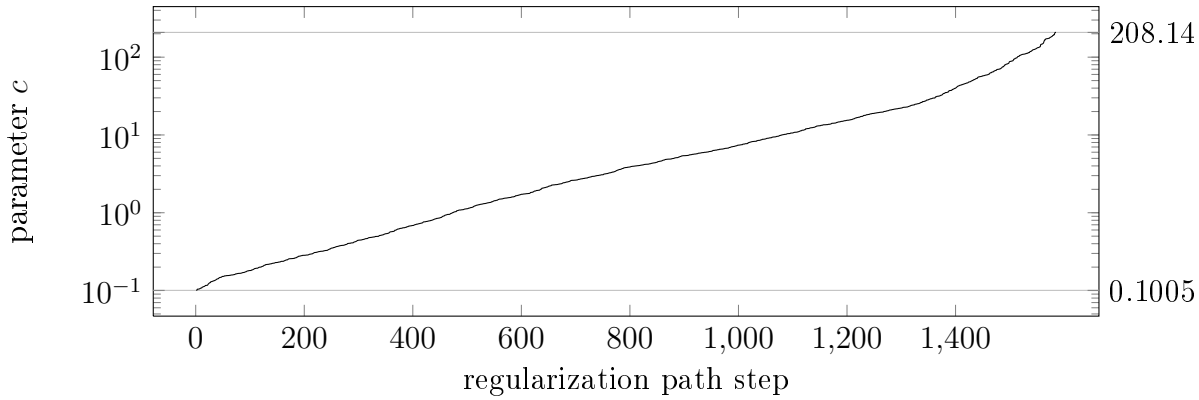

Figure 45: Adult | Gaussian Kernel | Regularization Parameter

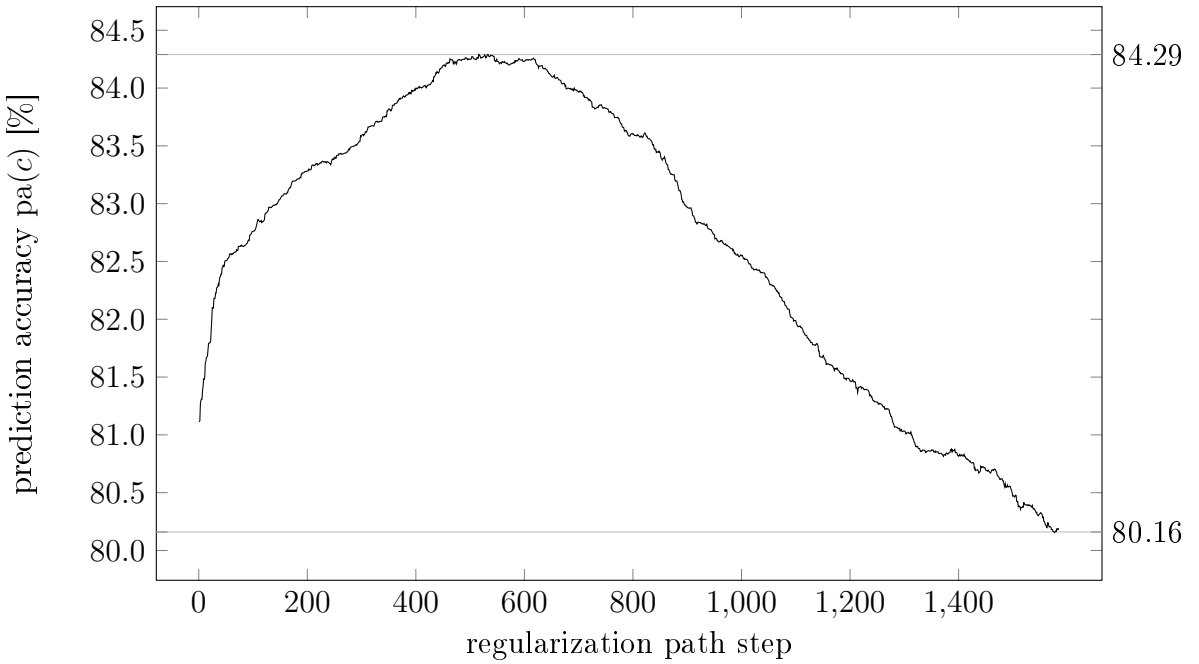

Figure 46: Adult | Gaussian Kernel | Prediction Accuracy

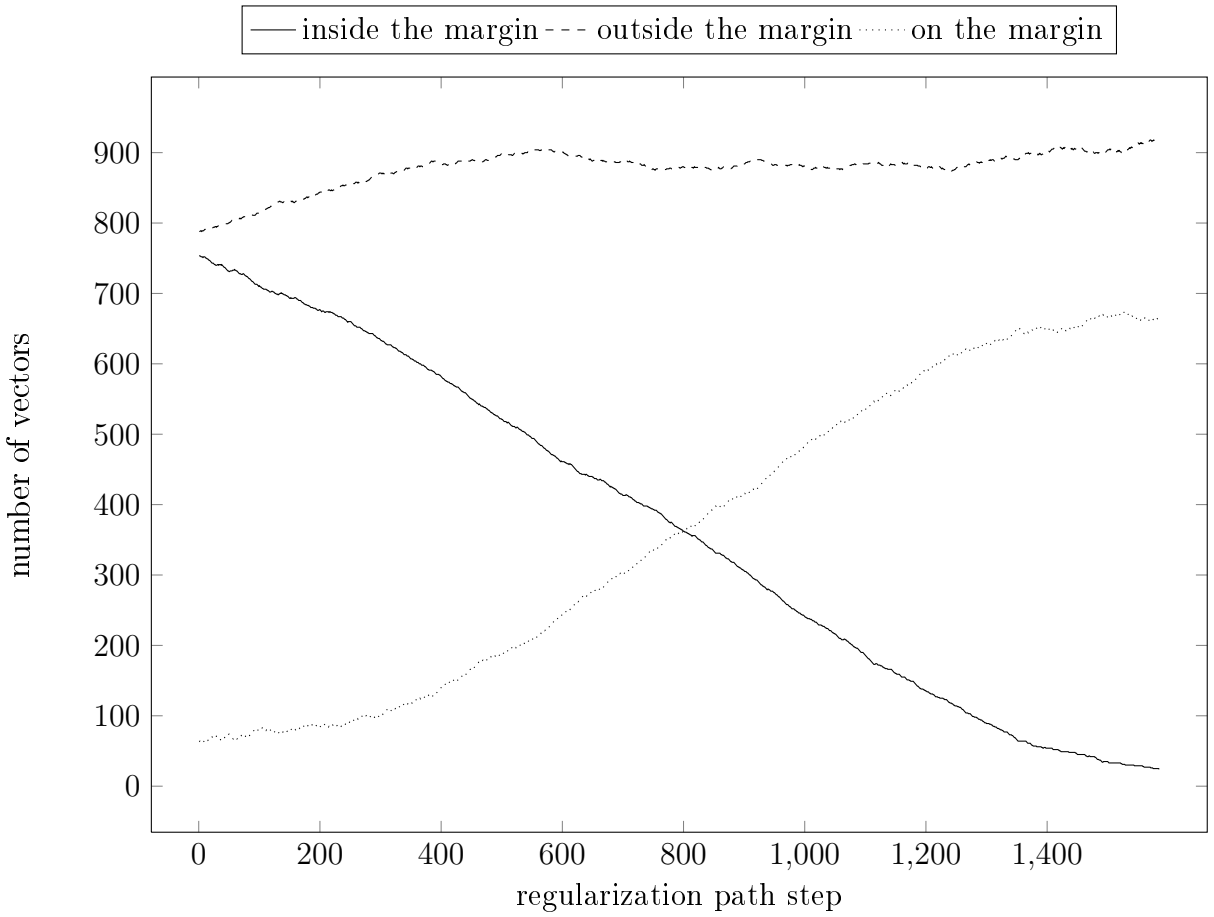

Figure 47: Adult | Gaussian Kernel | Vectors' Locations

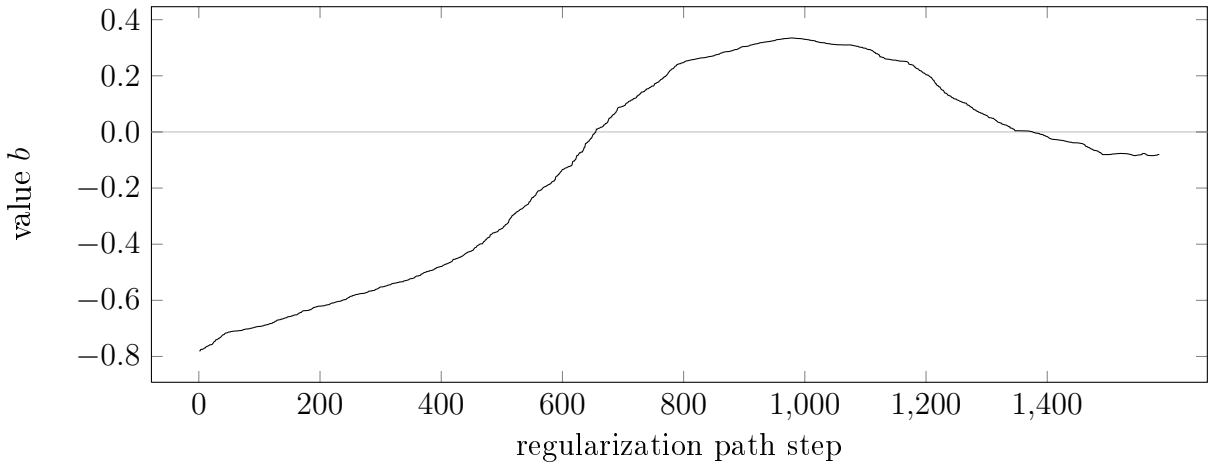

Figure 48: Adult | Gaussian Kernel | Value b

5.6.10 Splice Linear Kernel

$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{P}\rvert$	n	running time	path length	$\lvert B_0^*\rvert$	$\lvert\mathcal{T}_+\rvert$	$\lvert\mathcal{T}_-\rvert$
470	2175	60	17min	1072	31	228	242

Table 11: Splice data set overview using the linear kernel.

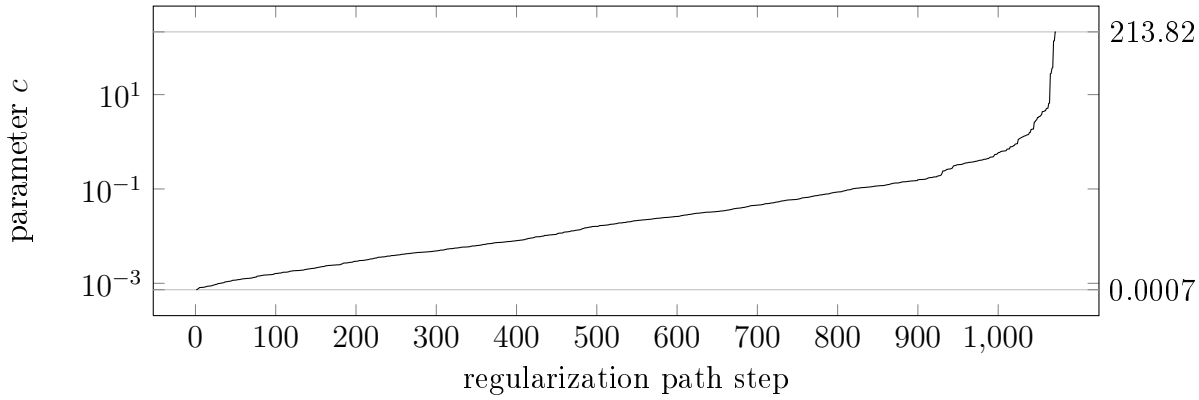

Figure 49: Splice | Linear Kernel | Regularization Parameter

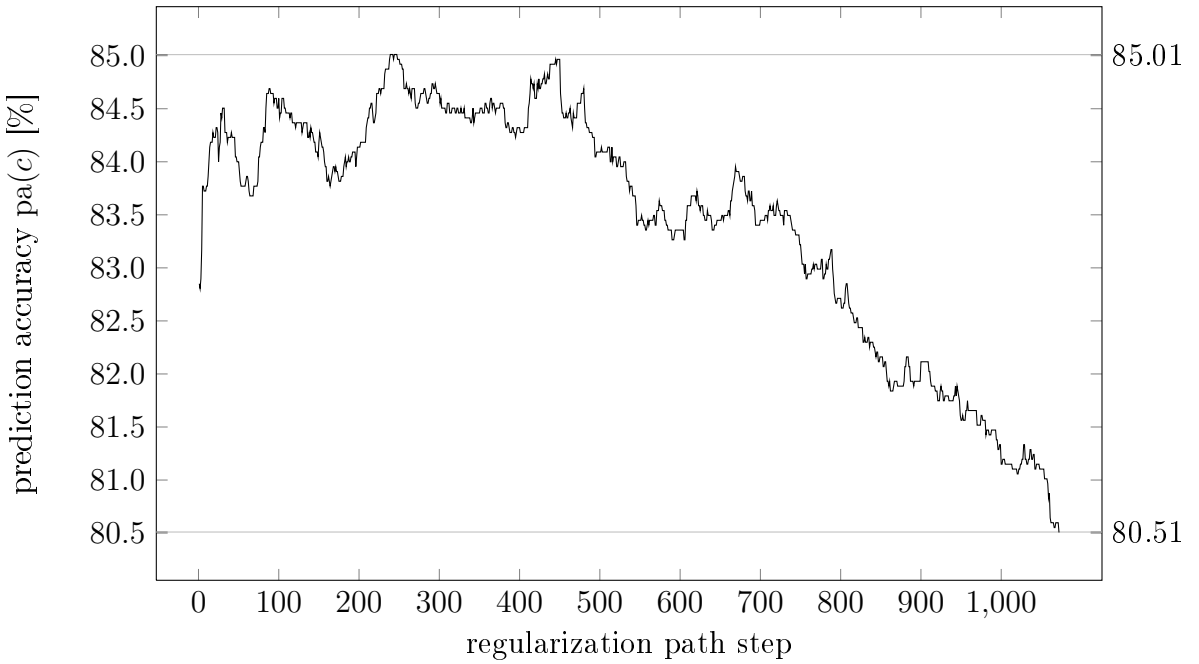

Figure 50: Splice | Linear Kernel | Prediction Accuracy

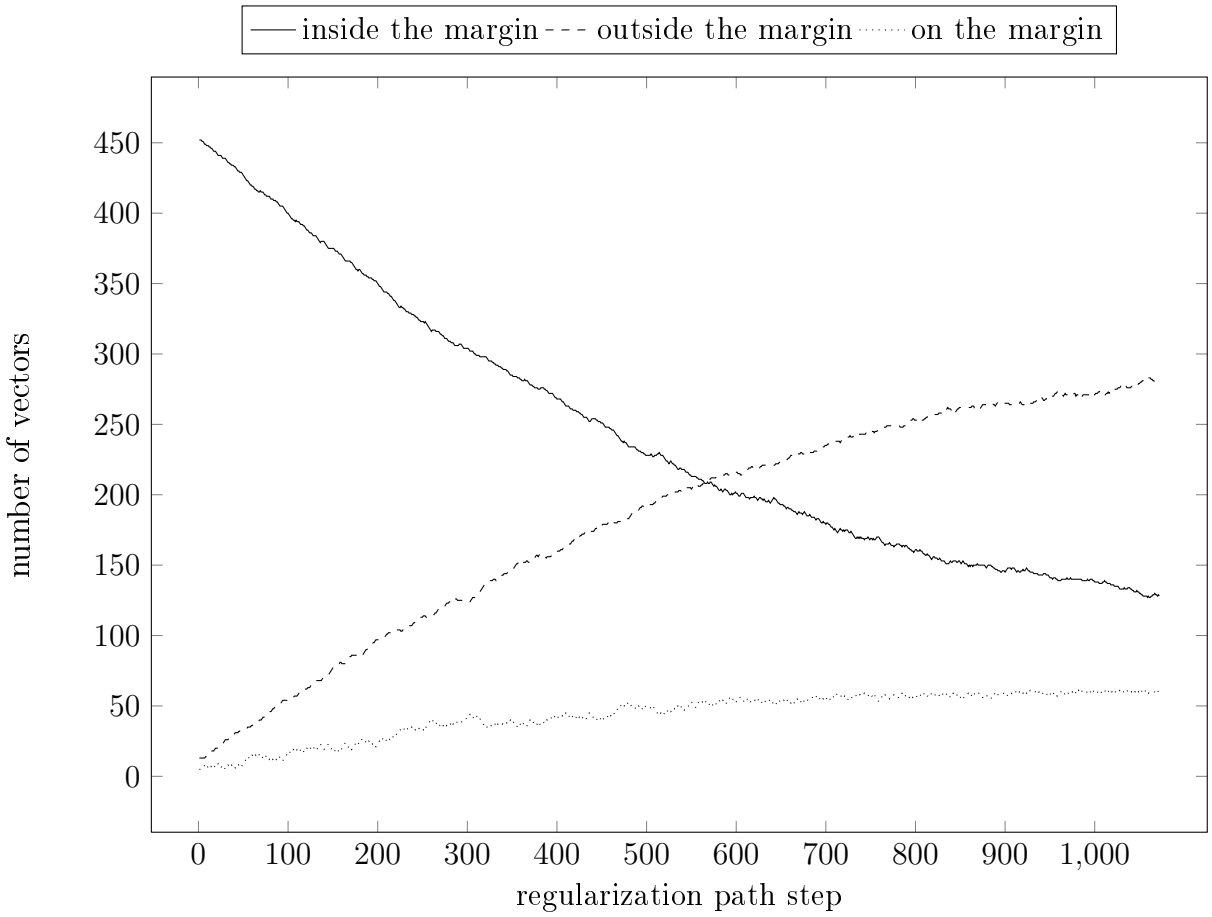

Figure 51: Splice | Linear Kernel | Vectors' Locations

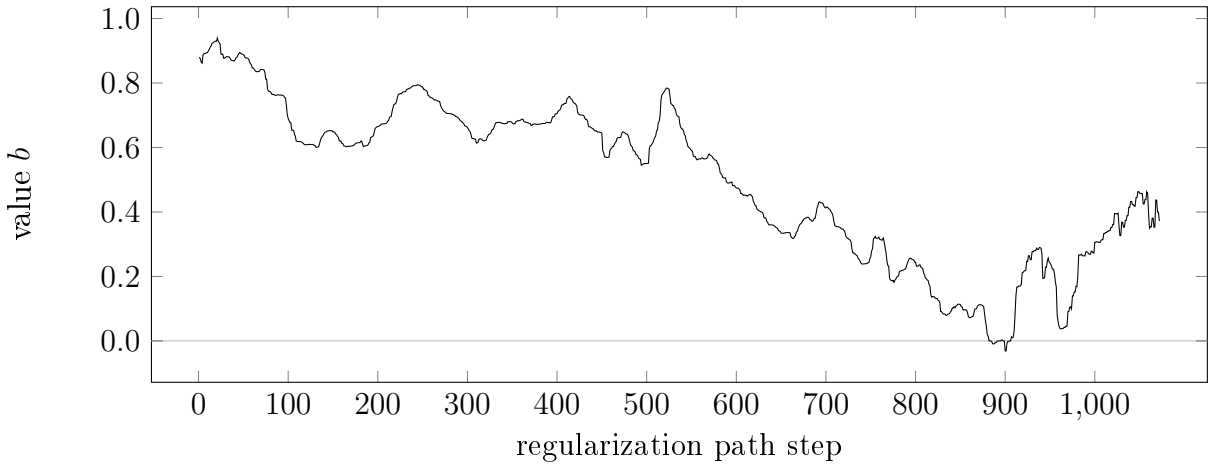

Figure 52: Splice | Linear Kernel | Value b

5.6.11 Splice Gaussian Kernel

$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{P}\rvert$	n	running time	path length	$\lvert B_0^*\rvert$	$\lvert\mathcal{T}_+\rvert$	$\lvert\mathcal{T}_-\rvert$
470	2175	60	19min	529	32	228	242

Table 12: Splice data set overview using the Gaussian Kernel.

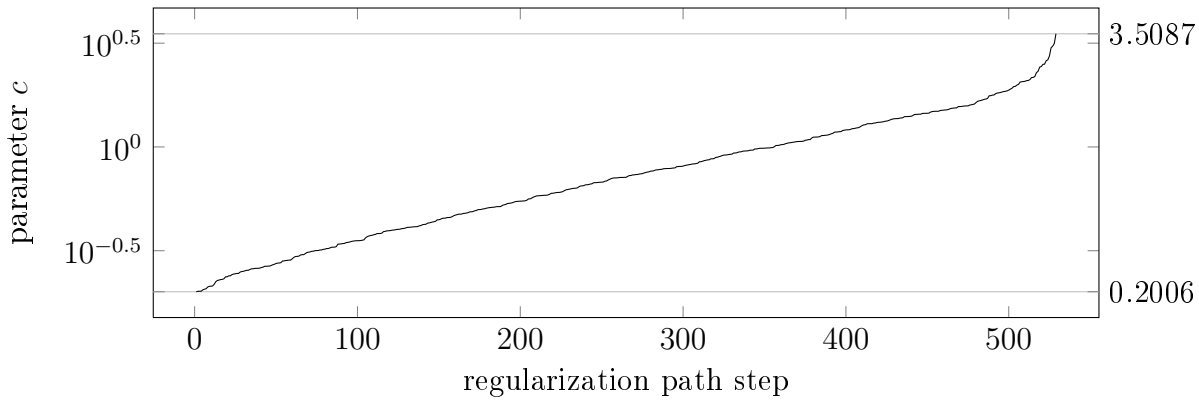

Figure 53: Splice | Gaussian Kernel | Regularization Parameter

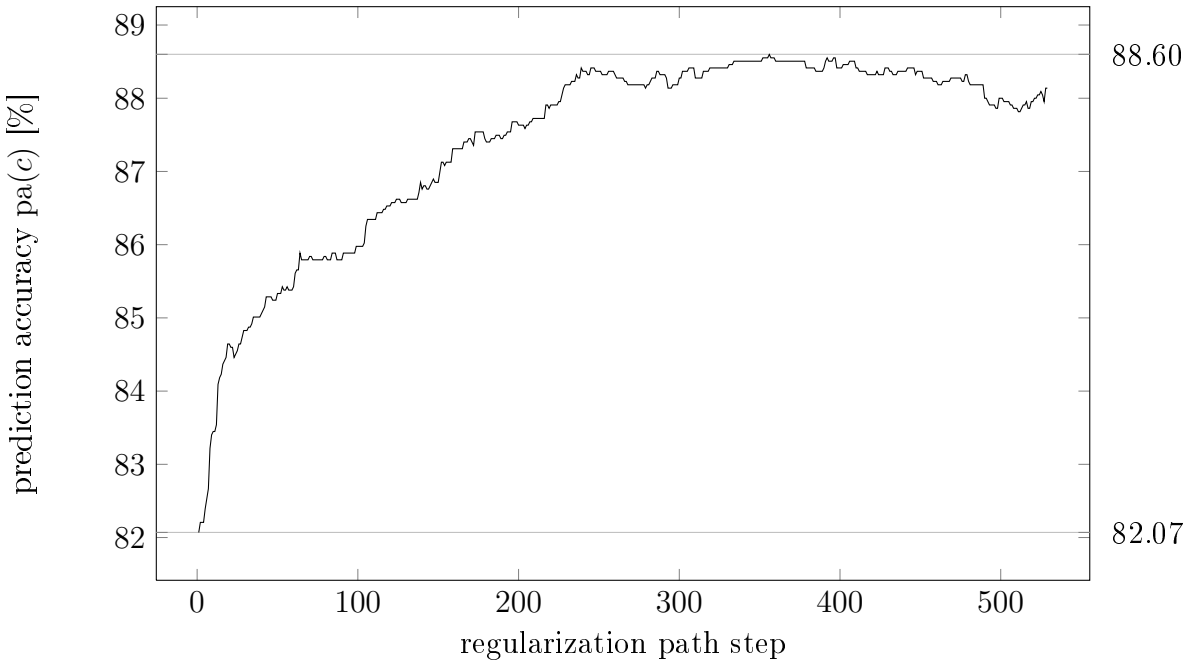

Figure 54: Splice | Gaussian Kernel | Prediction Accuracy

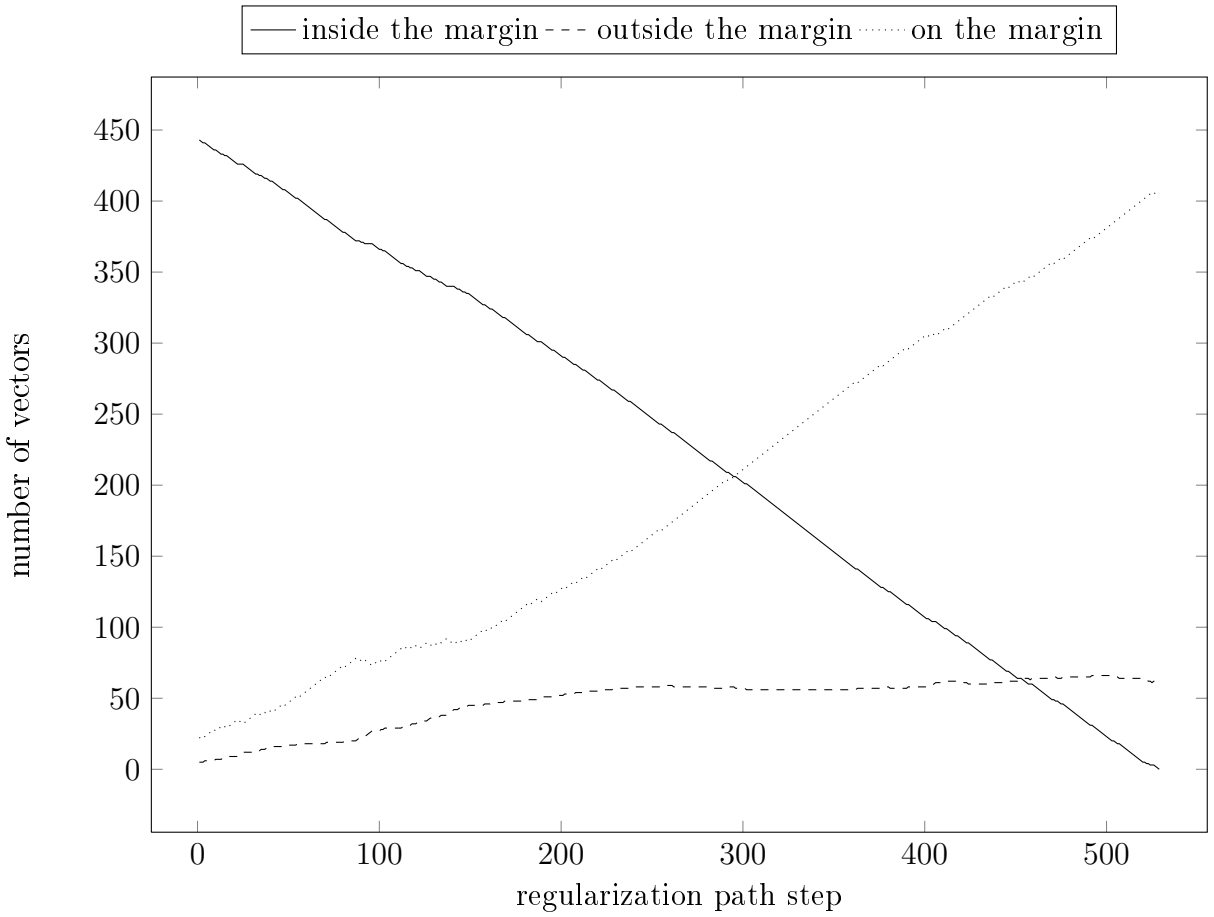

Figure 55: Splice | Gaussian Kernel | Vectors' Locations

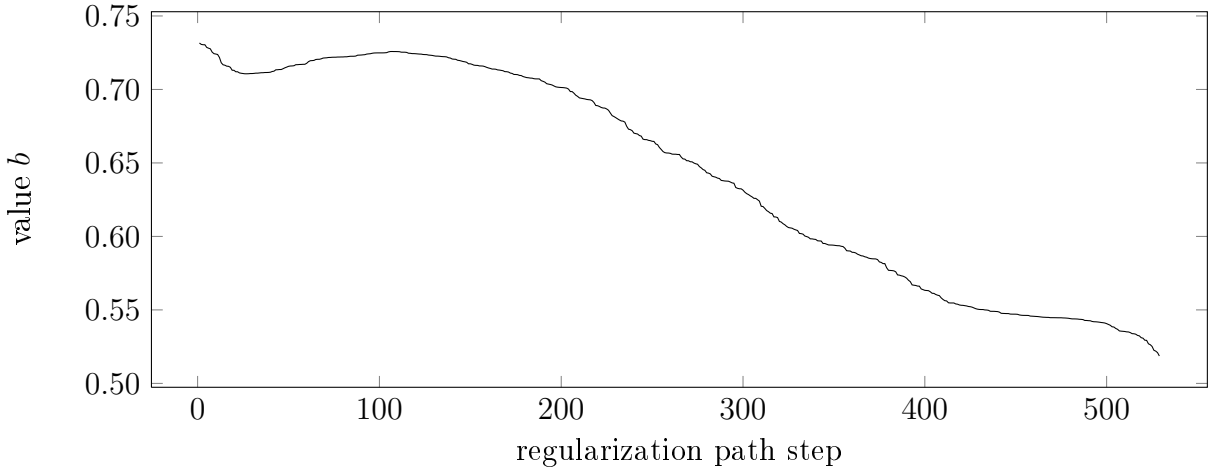

Figure 56: Splice | Gaussian Kernel | Value b

5.6.12 Simple

As mentioned in (5.6.1), we created this little SVM_{PSM} (4.2.2) example to demonstrate the processing of our implementation of the criss-cross method (4.1). In contrast to the other data sets, we recorded all **important results** of each step along the regularization path (4.1.4). To simplify the problem, we decided to use the linear kernel only.

The Simple data set is given by the following vectors, where the number i beside a vector refers to the position of $x^{(i)}$ in the training set $\mathcal{T}$ (4.2.1).

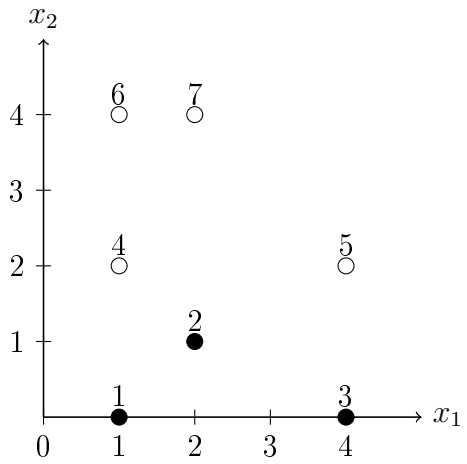

Figure 57: Overview of the Simple data set with two classes ∘ and •.

In the following, we visualize and describe the **geometrical changes** for each step according to (4.6) and the results in the two tables in the end of this subsection. Note that "ov" is the value of the objective function.

Steps 1 - 3: Starting with the vectors 6 and 7 on and all other vectors inside the margin, vector 6 moves outside the margin in step 2, whereas no change occurs in step 3 with respect to the vectors' locations. Meanwhile, the separating hyperplane (4.2.1) is given as follows according to w and b:

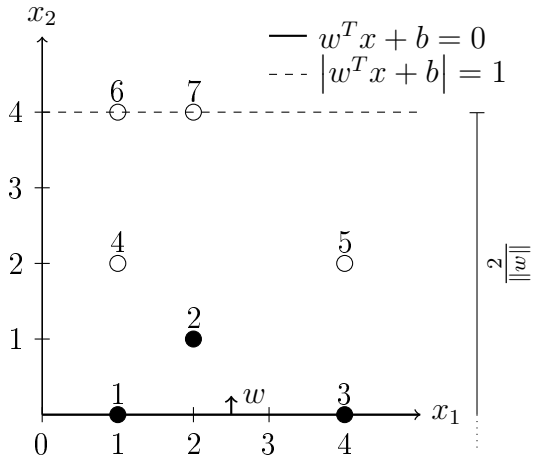

Figure 58: Steps 1 - 3 of the Simple SVM_{PSM} example.

Steps 4 - 6: The criss-cross method continues by moving the vectors 1 and 3 on the margin in steps 4 and 5. Afterwards, vector 7 moves outside the margin in step 6. Meanwhile, the separating hyperplane is given as follows according to w and b:

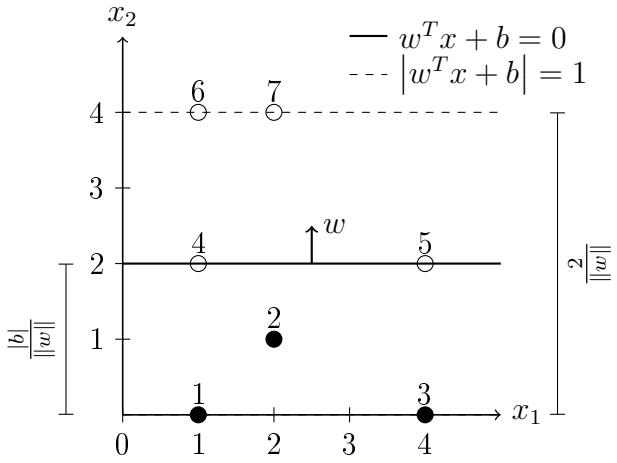

Figure 59: Steps 4 - 6 of the Simple SVM_{PSM} example.

Steps 7 - 9: According to the results in Table 10, vector 4 moves on the margin in step 7 followed by vector 5 after vector 1 moves outside the margin in step 8. In the end, vector 3 moves outside the margin in step 9. Note that in contrast to all other steps described, step 8 needed two rounds of the criss-cross algorithm (4.1.3) instead of one because $(\lambda_B^*(c))_{14}$ was considered as smaller than zero. Meanwhile, the separating hyperplane is given as follows according to w and b:

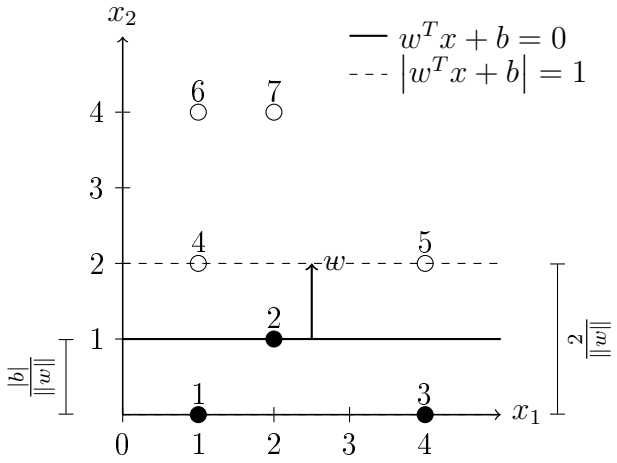

Figure 60: Steps 7 - 9 of the Simple SVM_{PSM} example.

Step 10: Finally, vector 2 moves on the margin in step 10 resulting in the separating hyperplane given as follows according to w and b:

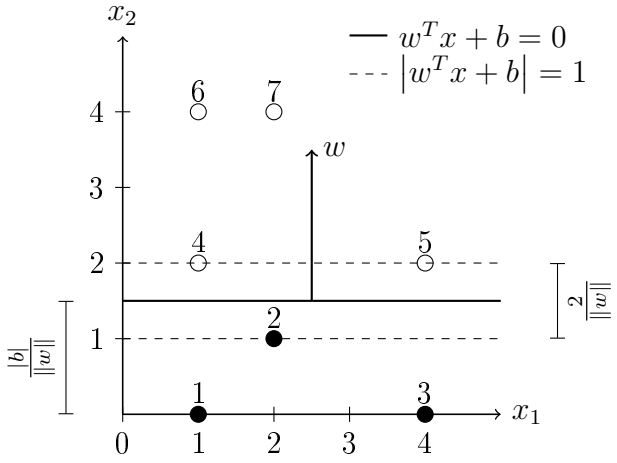

Figure 61: Step 10 of the Simple SVM_{PSM} example.

		B		B		B		B		B
step	1		2		3		4		5	
c	3.571E-02		3.571E-02		3.571E-02		7.143E-02		7.143E-02	
ov	-1.830E-01		1.830E-01		1.830E-01		3.036E-01		3.036E-01	
$\lambda^*_B(c)$	3.571E-02		3.571E-02		3.571E-02		7.143E-02		7.143E-02	
	3.571E-02		3.571E-02		3.571E-02		7.143E-02		7.143E-02	
	3.571E-02		3.571E-02		3.571E-02		7.143E-02		7.143E-02	
	3.571E-02		3.571E-02		3.571E-02		7.143E-02		7.143E-02	
	3.571E-02		3.571E-02		3.571E-02		7.143E-02		7.143E-02	
	0.000E+00		0.000E+00	X	0.000E+00	X	0.000E+00	X	0.000E+00	X
	3.571E-02		3.571E-02		3.571E-02		7.143E-02		7.143E-02	
	0.000E+00	X	0.000E+00	X	2.900E-06		1.000E+00		1.000E+00	
	0.000E+00	X	0.000E+00	X	0.000E+00	X	0.000E+00	X	0.000E+00	X
	1.000E+00		1.000E+00		1.000E+00		0.000E+00	X	0.000E+00	X
	1.250E+00		1.250E+00		1.250E+00		5.000E-01		5.000E-01	
	1.000E+00		1.000E+00		1.000E+00		0.000E+00		0.000E+00	X
	5.000E-01		5.000E-01		5.000E-01		1.000E+00		1.000E+00	
	5.000E-01		5.000E-01		5.000E-01		1.000E+00		1.000E+00	
	3.571E-02	X	3.571E-02	X	3.571E-02	X	7.143E-02	X	7.143E-02	X
	0.000E+00	X	0.000E+00	X	0.000E+00	X	0.000E+00	X	0.000E+00	X
b	0.000E+00		0.000E+00		0.000E+00		-1.000E+00		-1.000E+00	
w	1.250E-08		-9.000E-08		2.776E-17		-5.882E-09		0.000E+00	
	2.500E-01		2.500E-01		2.500E-01		5.000E-01		5.000E-01	
$\hat{\mu}$	2.857E-08		7.143E-08		3.571E-02		2.024E-08		9.524E-02	
r	6		8		10		12		7	

Table 13: Part 1: Criss-cross algorithm's results along the regularization path of the Simple data set.

		B		B		B		B		B
step	6		7		8		9		10	
c	1.667E-01		3.333E-01		4.286E-01		1.000E+00		2.000E+00	
ov	5.417E-01		8.333E-01		9.286E-01		1.500E+00		2.000E+00	
$\lambda^*_B(c)$	5.556E-02		1.111E-01		0.00E+00	X	4.000E-07	X	2.000E+00	X
	1.667E-01		3.333E-01		4.286E-01		1.000E+00		2.000E+00	
	1.111E-01		2.222E-01		2.857E-01		4.000E-07	X	2.000E+00	X
	1.667E-01		3.333E-01		2.857E-01		6.667E-01		1.333E+00	
	1.667E-01		3.333E-01		4.286E-01		3.333E-01		6.667E-01	
	1.600E-06	X	2.000E+00	X	2.000E+00	X	2.000E+00	X	4.000E+00	X
	1.600E-06	X	2.000E+00	X	2.000E+00	X	2.000E+00	X	4.000E+00	X
	1.000E+00		1.000E+00		1.000E+00		1.000E+00		3.000E+00	
	0.000E+00	X	0.000E+00	X	0.000E+00	X	0.00E+00	X	0.000E+00	X
	1.111E-01	X	2.222E-01	X	4.286E-01	X	1.000E+00	X	2.000E+00	X
	5.000E-01		1.000E+00		1.000E+00		1.000E+00		0.00E+00	X
	5.556E-02	X	1.111E-01	X	1.429E-01	X	1.000E+00	X	2.000E+00	X
	1.000E+00		0.000E+00	X	1.429E-01	X	3.333E-01	X	6.667E-01	X
	1.000E+00		0.000E+00		1.000E-07	X	6.667E-01	X	1.333E+00	X
	1.667E-01	X	3.333E-01	X	4.286E-01	X	1.000E+00	X	2.000E+00	X
	1.667E-01	X	3.333E-01	X	4.286E-01	X	1.000E+00	X	2.000E+00	X
b	-1.000E+00		-1.000E+00		-1.000E+00		-1.000E+00		-3.000E+00	
w	0.000E+00		0.000E+00		0.000E+00		2.22E-16		0.00E+00	
	5.000E-01		1.000E+00		1.000E+00		1.0000002		2.00E+00	
$\hat{\mu}$	1.667E-01		9.524E-02		5.714E-01		1.000E+00			
r	13		1		3		11			

Table 14: Part 2: Criss-cross algorithm's results along the regularization path of the Simple data set.

6 Statement of Independence

I confirm that I did the work at hand independently and have been using the indicated sources and aids only.

Weimar, March 2011

Torsten Welsch

7 References

[Abe05] S. Abe. Support Vector Machines for Pattern Classification. *Springer*, 2005.

[AG75] I. Adler and D. Gale. On the solutions of the positive semi-definite complementarity problem. *Technical Report*, 1975.

[BB09] D. Baier and M. Brusch. Conjointanalyse. *Springer*, 2009.

[BEPW08] K. Backhaus, B. Erichson, W. Plinke, and R Weiber. Multivariante Analysemethoden. *Springer*, 2008.

[BG69] R. H. Bartels and G. H. Golub. The simplex method of linear programming using LU decomposition. *Communications of the ACM*, 12:266–268, 1969.

[CCCJ01] C. Chih-Chung and L. Chih-Jen. LIBSVM: a library for support vector machines. 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[CPS92] R. Cottle, J. S. Pang, and R. E. Stone. The Linear Complementarity Problem. *Academic Press*, 1992.

[FNT98] K. Fukuda, M. Namiki, and A. Tamura. EP theorems and linear complementarity problems. *Discrete Applied Mathematics*, 84:107–119, 1998.

[FT72] J. J. H. Forrest and J. A. Tomlin. Updated triangular factors of the basis to maintain sparsity in the product form simplex method. *Mathematical Programming*, 2:263–278, 1972.

[Her06] M. Hermann. Numerische Mathematik. *Oldenbourg*, 2006.

[Hig08] N. J. Higham. Functions of Matrices: Theory and Computation. *SIAM*, 2008.

[IBM09] IBM. IBM ILOG CPLEX V12.1: File formats supported by CPLEX. 2009. Available at ftp://ftp.software.ibm.com/software/websphere/ilog/docs/optimization/cplex/ps_reffileformatscplex.pdf (14.03.2011).

[JMT04] H. Th. Jongen, K. Meer, and E. Triesch. Optimization Theory. *Kluwer Academic Publishers*, 2004.

[KT92] E. Klafszky and T. Terlaky. Some generalizations of the criss-cross method for quadratic programming. *Optimization*, 24:127–139, 1992.

[LS90] D. Lloyd Smith. Mathematical Programming Methods in Structural Plasticity. *Springer*, 1990.

[Rei82] J. K. Reid. A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases. *Mathematical Programming*, 24:55–69, 1982.

[Rit62] K. Ritter. Ein Verfahren zur Lösung parameter-abhängiger, nicht-linearer Maximum-Probleme. *Unternehmensforschung*, 6:149–166, 1962.

[SK06] H. R. Schwarz and N. Köckler. Numerische Mathematik. *Teubner*, 2006.

[SS90] U. H. Suhl and L. M. Suhl. Computing Sparse LU Factorization for Large-Scale Linear Programming Bases. *ORSA Journal on Computing*, 2:325–335, 1990.

[SS93] U. H. Suhl and L. M. Suhl. A fast LU update for linear programming. *Annals of Operations Research*, 43:33–47, 1993.

[SS02] B. Schölkopf and A. J. Smola. Learning with Kernels. *The MIT Press*, 2002.

[VC95] V. Vapnik and C. Cortes. Support-Vector Networks. *Springer Machine Learning*, 20:273–297, 1995.

[Wan05] L. (Ed.) Wang. Support Vector Machines: Theory and Applications. *Springer*, 2005.

[Wel10] T. Welsch. An Exact pQP-Solver Implementation. 2010.

8 List of Figures

1 Simple QP example in which $\frac{1}{2}x_1^2 + x_2^2 - 8x_2$ defines the objective function and $(\frac{4}{9}, \frac{28}{9})^T$ is the optimal solution ∘. 7
2 Simple SVM example (5.6.12) in which $w^T = (0, 2)^T$ and $b = -3$ define the separating hyperplane of the two classes ∘ and •. 15
3 Illustration of mapping two linearly non-separable classes ∘ and • into a higher-dimensional feature space where they become linearly separable. . . 19
4 Columnwise data structure of matrix M. 43
5 Rowwise data structure of matrix M_B. 44
6 Columnwise data structure of matrix M_B. 44
7 Data structure of row/column indices according to their number of non-zero values in the active submatrix. 45

8 Permutation vectors' data structure. 46
9 Step 1: solving $L \cdot z = Q \cdot q = b$. 48
10 Step 2: solving $U \cdot R^{-1} \cdot \lambda_B = U \cdot x = z$. 49
11 Example of the LIBSVM data format recording a training set $\mathcal{T}$ (4.2.1). . . 57
12 Example of the CPLEX LP file format recording an SVM_{DSM}-pQP. 58
13 Example of the SOL file format recording the solution of an SVM_{DSM}-pQP. 58
14 Engine | Linear Kernel | Regularization Parameter 65
15 Engine | Linear Kernel | Prediction Accuracy 65
16 Engine | Linear Kernel | Vectors' Locations 66
17 Engine | Linear Kernel | Value b 66
18 Engine | Linear Kernel | Part Worth Values 67
19 Engine | Gaussian Kernel | Regularization Parameter 68
20 Engine | Gaussian Kernel | Prediction Accuracy 68
21 Engine | Gaussian Kernel | Vectors' Locations 69
22 Engine | Gaussian Kernel | Value b 69
23 CeBIT1 | Linear Kernel | Regularization Parameter 70
24 CeBIT1 | Linear Kernel | Prediction Accuracy 70
25 CeBIT1 | Linear Kernel | Vectors' Locations 71
26 CeBIT1 | Linear Kernel | Value b 71
27 CeBIT1 | Linear Kernel | Part Worth Values 72
28 CeBIT1 | Gaussian Kernel | Regularization Parameter 73
29 CeBIT1 | Gaussian Kernel | Prediction Accuracy 73
30 CeBIT1 | Gaussian Kernel | Vectors' Locations 74
31 CeBIT1 | Gaussian Kernel | Value b 74
32 CeBIT2 | Linear Kernel | Regularization Parameter 75
33 CeBIT2 | Linear Kernel | Prediction Accuracy 75
34 CeBIT2 | Linear Kernel | Vectors' Locations 76
35 CeBIT2 | Linear Kernel | Value b 76
36 CeBIT2 | Linear Kernel | Part Worth Values 77
37 CeBIT2 | Gaussian Kernel | Regularization Parameter 78
38 CeBIT2 | Gaussian Kernel | Prediction Accuracy 78
39 CeBIT2 | Gaussian Kernel | Vectors' Locations 79
40 CeBIT2 | Gaussian Kernel | Value b 79
41 Adult | Linear Kernel | Regularization Parameter 80
42 Adult | Linear Kernel | Prediction Accuracy 80
43 Adult | Linear Kernel | Vectors' Locations 81
44 Adult | Linear Kernel | Value b 81
45 Adult | Gaussian Kernel | Regularization Parameter 82
46 Adult | Gaussian Kernel | Prediction Accuracy 82
47 Adult | Gaussian Kernel | Vectors' Locations 83
48 Adult | Gaussian Kernel | Value b 83
49 Splice | Linear Kernel | Regularization Parameter 84
50 Splice | Linear Kernel | Prediction Accuracy 84

51 Splice | Linear Kernel | Vectors' Locations 85
52 Splice | Linear Kernel | Value b 85
53 Splice | Gaussian Kernel | Regularization Parameter 86
54 Splice | Gaussian Kernel | Prediction Accuracy 86
55 Splice | Gaussian Kernel | Vectors' Locations 87
56 Splice | Gaussian Kernel | Value b 87
57 Overview of the Simple data set with two classes ∘ and •. 88
58 Steps 1 - 3 of the Simple SVM_{PSM} example. 89
59 Steps 4 - 6 of the Simple SVM_{PSM} example. 89
60 Steps 7 - 9 of the Simple SVM_{PSM} example. 90
61 Step 10 of the Simple SVM_{PSM} example. 90

9 List of Tables

1 Correctly determined parameter c_0. 29
2 Incorrectly determined parameter c_0. 29
3 Engine data set overview using the linear kernel. 65
4 Engine data set overview using the Gaussian kernel. 68
5 CeBIT1 data set overview using the linear kernel. 70
6 CeBIT1 data set overview using the Gaussian kernel. 73
7 CeBIT2 data set overview using the linear kernel. 75
8 CeBIT2 data set overview using the Gaussian kernel. 78
9 Adult data set overview using the linear kernel. 80
10 Adult data set overview using the Gaussian kernel. 82
11 Splice data set overview using the linear kernel. 84
12 Splice data set overview using the Gaussian Kernel. 86
13 Part 1: Criss-cross algorithm's results along the regularization path of the Simple data set. 91
14 Part 2: Criss-cross algorithm's results along the regularization path of the Simple data set. 92